Genesis of Intrusive-Related Hydrothermal Gold Deposits

Appendix

THE AUSTRALIAN
LIBRARY
NATIONAL UNIVERSITY

Contents

A Electron Microprobe Analyses **1**

B Abbreviations and Sample Catalog **9**

C Thermodynamic Calculations **14**
C.1 Standard State Properties and Equation of State Parameters for Gold Complexes 14
C.2 Reaction Progress Calculations 15

D Source Code of the Monte Carlo Method **17**

E C-Source Code of the Fluid Tracking Method **44**

F Implementation of the Electric Analog Model **71**
F.1 Hardware Setup and Technical Specifications 71
F.2 C-Source Code for Data Aquisition 74
F.3 Preparation of Electric Analog Models 95
F.4 Example of Scaling of an Electric Analog Model 96

List of Tables

A.1 Microprobe analyses of almandine and muscovite from Plateau Point (1) . . . 2
A.2 Microprobe analyses from hornfels at Plateau Point (2) . . . 2
A.3 Microprobe analyses of muscovite from hornfels at Hayes Creek . . . 3
A.4 Microprobe analyses of muscovite from hornfels 2.2 km east of Hayes Creek . . . 3
A.5 Microprobe analyses of muscovite from quartz vein selvage 15 km south of Bridge Creek . . . 4
A.6 Microprobe analyses of biotite from biotite-schist from the Mount Shoobridge Gold and Tin Mine . . . 4
A.7 Microprobe analyses of garnet and biotite from gold-vein selvage from the Zapopan Prospect . . . 5
A.8 Microprobe analyses from the Zapopan Prospect and the Woolwonga Gold Mine . . . 5
A.9 Microprobe analyses of garnet and biotite from gold-mineralized hornfels from the Woolwonga Gold Mine (1) . . . 6
A.10 Microprobe analyses from gold-mineralized hornfels from the Woolwonga Gold Mine (2) . . . 6
A.11 K-spar and biotite analyses from Au-vein selvage from the Enterprise Gold Mine . . . 7
A.12 Microprobe analyses of actinolite from a carbonaceous gold-vein selvage at the Woolwonga Gold Mine . . . 7
A.13 Microprobe analyses of biotite from hornfels of the Phantom Pit at Cosmo Howley . . . 8
A.14 Microprobe analyses of garnet II and biotite from Cosmo Howley . . . 8

B.1 Abbreviations used in sample descriptions in tables . . . 10
B.2 Storage of samples referred to in Chapter 1 . . . 11
B.3 Storage of samples referred to in Chapter 2 . . . 11
B.4 Storage of samples referred to in Chapter 3 . . . 11
B.5 Storage of samples referred to in Chapter 4 . . . 12
B.6 Storage of samples referred to in Chapter 5 . . . 13

C.1 Fitting parameters for gold complexes . . . 15
C.2 Composition of gold-bearing fluid at 550°C, 200 MPa . . . 16

List of Figures

F.1 Data-aquisition hardware set-up of for the electric analog modelling . 73

Appendix A

Electron Microprobe Analyses

93198	G1	G2	G3	G4	G5	Ave.	M1	M2	M3	M4
SiO_2	36.31	36.32	36.69	36.59	36.63	36.51	45.15	45.05	44.83	45.22
TiO_2	<	<	<	<	<	<	<	0.18	<	0.10
Al_2O_3	20.53	20.47	20.9	20.74	20.68	20.66	35.83	35.75	36.26	35.86
FeO	40.60	40.34	40.94	40.39	41.11	40.68	1.54	1.26	1.34	1.46
MnO	1.78	2.09	1.85	1.98	1.72	1.89	<	<	<	<
MgO	<	<	0.11	<	<	0.02	<	<	<	<
CaO	0.57	0.74	0.56	0.47	0.52	0.57	<	<	<	<
Na_2O	<	<	<	<	<	<	0.17	<	0.16	0.16
K_2O	<	<	<	<	<	<	10.86	11.02	10.76	10.99
Total	99.79	99.97	101.05	100.18	100.65	100.33	93.55	93.26	93.35	93.79
n						6				
Si p.f.u.							3.06	3.06	3.03	3.05

Table A.1: Microprobe analyses of almandine and muscovite from hornfels at Plateau Point 1 (Sample 93198). In all analyses including the following tables, the detection limits in wt.% are: SiO_2 = 0.07, TiO_2 = 0.06, Al_2O_3, FeO = 0.08, MnO = 0.06, MgO = 0.07, CaO = 0.06, Na_2O = 0.01, K_2O = 0.05; SO_3, F, Cl in ppm: 114, 43, 43, respectively. G = garnet, M = muscovite. In all tables n is the number of individual analyses. Ave. = average.

93198	M5	M6	M7	M8	Ave.
SiO_2	44.72	45.22	45.73	45.45	45.171
TiO_2	<	<	<	<	0.04
Al_2O_3	35.86	36.47	36.55	37.08	36.21
FeO	1.75	0.76	0.85	0.96	1.24
MnO	<	<	<	<	<
MgO	<	<	<	<	<
CaO	<	<	<	<	<
Na_2O	<	<	<	<	0.06
K_2O	10.96	10.79	10.93	11.00	10.91
Total	93.29	93.24	94.06	94.49	93.63
n					8
Si p.f.u.	3.03	3.05	3.07	3.03	3.05

Table A.2: Microprobe analyses from hornfels at Plateau Point 2 (Sample 93198). M = muscovite.

93226	1	2	3	4	5	6	7	8	Ave.
SiO_2	48.8	46.74	45.58	45.98	46.00	48.00	46.01	46.08	46.65
TiO_2	0.13	<	0.19	0.09	0.38	0.16	<	<	0.12
Al_2O_3	37.17	32.62	34.80	35.47	35.18	33.58	35.31	35.37	34.94
FeO	0.90	1.80	0.82	1.01	0.96	1.22	1.15	0.92	1.10
MnO	<	<	<	<	<	<	<	<	<
MgO	<	0.91	0.15	0.15	<	0.50	<	0.16	0.23
CaO	<	<	<	<	<	<	<	<	<
Na_2O	<	<	<	<	<	<	<	<	<
K_2O	11.67	11.10	10.80	11.07	10.97	10.83	11.16	11.20	11.10
Total	98.67	93.17	92.33	93.78	93.48	94.29	93.63	93.73	94.14
n									8
Si p.f.u.	3.13	3.18	3.12	3.10	3.12	3.22	3.12	3.11	3.14

Table A.3: Microprobe analyses of muscovite from hornfels at Hayes Creek (Sample 93226).

93231	1	2	3	4	5	6	7	8	Ave.
SiO_2	45.01	45.29	45.85	46.01	45.32	45.63	45.05	45.68	45.48
TiO_2	0.33	0.78	1.00	0.84	0.53	0.31	0.94	0.10	0.60
Al_2O_3	33.8	34.1	34.01	34.48	33.99	35.97	33.89	35.03	34.41
FeO	2.85	2.69	2.97	2.75	2.96	1.75	2.90	2.91	2.72
MnO	<	<	<	<	<	<	<	<	<
MgO	0.48	0.47	0.56	0.51	0.55	0.4	0.5	0.57	0.51
CaO	<	<	<	<	<	<	<	<	<
Na_2O	0.30	0.28	0.29	0.26	0.31	0.24	0.33	0.22	0.28
K_2O	9.15	9.27	9.49	8.58	9.03	8.98	10.20	8.88	9.20
SO_3	<	211	<	<	<	<	<	189	<
F	203	<	<	249	997	408	643	564	383
Cl	<	<	<	<	87	72	47	<	<
Total	91.94	92.93	94.17	93.45	92.81	93.34	93.89	93.48	93.25
n									8
Si p.f.u.	3.06	3.05	3.06	3.06	3.05	3.03	3.04	3.04	3.05

Table A.4: Microprobe analyses of muscovite from hornfels 2.2 km east of Hayes Creek (Sample 93231).

93235	M1	M2	M3	M4	M5	Ave.	K1	A1
SiO_2	47.56	47.52	47.34	46.42	45.54	46.88	74.24	68.46
TiO_2	0.50	<	0.21	<	0.21	0.18	<	<
Al_2O_3	32.99	34.42	34.92	35.36	34.84	34.51	13.92	19.98
FeO	1.38	1.11	1.83	1.21	1.15	1.34	<	0.18
MnO	<	<	<	<	<	<	<	<
MgO	0.88	0.47	<	<	<	0.27	<	<
CaO	<	<	<	<	<	<	<	0.46
Na_2O	<	<	<	<	<	<	0.85	9.34
K_2O	11.24	11.32	11.17	11.08	11.07	11.18	11.95	1.49
Total	94.55	94.84	95.47	94.07	92.82	94.35	100.96	99.91
n						5		
Si p.f.u.	3.20	3.19	3.15	3.13	3.11	3.16		

Table A.5: Microprobe analyses of muscovite from quartz-vein selvage 15 km south of Bridge Creek (Sample 93235). M = muscovite, K = K-feldspar, A = albite.

93245	1	2	3	4	5	6	7	8	Ave.
SiO_2	40.53	39.78	43.63	39.17	40.69	43.9	44.12	44.55	42.05
TiO_2	0.55	0.68	0.41	0.70	0.46	0.40	0.40	0.41	0.50
Al_2O_3	22.44	22.70	27.17	22.33	22.30	26.88	27.58	27.82	24.90
FeO	12.62	14.00	6.84	14.17	13.20	7.12	6.43	6.49	10.11
MnO	0.38	0.48	0.29	0.53	0.39	0.20	0.26	0.26	0.35
MgO	2.97	2.93	2.56	2.97	3.08	2.50	2.26	2.27	2.69
CaO	<	<	<	<	<	<	<	<	<
Na_2O	0.22	0.21	0.27	0.19	0.19	0.24	0.30	0.33	0.24
K_2O	9.06	8.39	8.89	8.61	9.01	8.98	9.29	9.38	8.95
SO_3	<	120	<	343	149	<	<	<	76
F	54254	55291	44026	53682	56261	44507	40862	42843	48966
Cl	75	50	47	70	88	<	<	<	<
Total	94.21	94.74	94.46	94.13	95.00	94.67	94.73	95.80	94.72
n									8

Table A.6: Microprobe analyses of biotite from biotite-schist from the Mount Shoobridge Gold and Tin Mine (Sample 93245).

93267	G1	G2	G3	G4	Ave.	B1	B2	B3	B4	Ave.	K1
SiO_2	36.73	36.85	37.14	37.23	36.99	34.12	33.29	33.9	32.99	33.58	61.11
TiO_2	<	<	<	0.09	0.02	0.71	0.74	0.37	0.35	0.54	<
Al_2O_3	20.13	20.21	20.15	20.09	20.15	18.24	18.21	18.85	18.17	18.37	18.41
FeO	18.80	18.88	21.37	21.06	20.03	26.45	26.66	26.13	27.20	26.61	2.67
MnO	16.06	16.14	10.67	10.08	13.24	0.13	0.22	0.19	0.21	0.19	<
MgO	0.09	0.09	0.22	0.34	0.19	5.28	5.44	5.52	5.54	5.45	0.30
CaO	6.61	6.64	9.77	9.94	8.24	<	<	<	<	<	<
Na_2O	<	<	<	<	<	0.05	0.05	0.04	0.06	0.05	0.49
K_2O	<	<	<	<	<	9.45	8.41	9.37	8.12	8.84	14.01
SO_3	<	176	<	<	<	161	331	305	976	443	<
F	<	<	<	<	<	3444	3876	3751	6152	4306	<
Cl	84	<	56	<	35	426	438	361	351	394	<
Total	98.43	98.86	99.33	98.86	98.87	94.86	93.53	94.85	93.53	94.19	96.99
n					4					4	

Table A.7: Microprobe analyses of garnet and biotite from gold-vein selvage from the Zapopan Prospect. G = garnet, B = biotite, K = K-feldspar.

	B1	G1	G2	K1	K2	M1
SiO_2	33.17	36.02	35.79	64.82	64.12	46.41
TiO_2	0.39	<	<	<	<	<
Al_2O_3	18.42	20.32	19.81	17.88	17.93	33.69
FeO	29.86	17.96	21.21	0.13	<	3.17
MnO	<	19.89	15.70	<	<	<
MgO	5.40	<	<	<	<	0.40
CaO	<	5.97	6.64	<	<	<
Na_2O	<	<	<	<	<	<
K_2O	7.93	<	<	16.93	17.21	9.43
Total	95.16	100.16	99.15	99.77	99.26	93.11
Si p.f.u.						3.12

Table A.8: Microprobe analyses from the Zapopan Prospect (B1, G1, G2, Sample 93267) and the Woolwonga Gold Mine (K1, K2, M1, Sample 93415). G = garnet, B = biotite, K = K-feldspar.

93271	G1	G2	G3	G4	G5	B1	B2	B3	B4	B5	B-Ave.
SiO_2	37.10	36.97	37.45	36.15	36.17	34.45	34.07	31.04	33.76	33.38	33.34
TiO_2	<	<	<	<	<	1.01	1.31	0.62	1.33	1.26	1.11
Al_2O_3	21.10	20.87	21.16	20.78	20.81	18.45	18.65	18.38	18.76	18.74	18.59
FeO	17.35	18.01	16.72	16.77	17.35	22.78	23.63	24.86	23.49	23.37	23.63
MnO	23.13	22.82	24.33	23.42	22.98	0.69	0.68	0.53	0.74	0.63	0.65
MgO	0.24	0.18	0.14	0.20	0.17	8.03	7.78	11.08	8.21	8.21	8.66
CaO	2.83	2.81	2.90	2.49	2.78	<	<	<	<	<	<
Na_2O	0.22	<	<	<	<	0.11	0.07	<	<	<	0.04
K_2O	<	<	<	<	<	8.84	8.85	4.21	8.12	7.81	7.57
SO_3	<	<	<	<	<	257	329	<	<	<	117
F	<	<	<	<	<	5517	5600	4182	4473	5479	5050
Cl	<	<	<	<	<	1983	2237	1030	2008	2091	1870
Total	101.97	101.67	102.7	99.81	100.26	95.17	95.91	91.28	95.13	94.22	94.34
n											5

Table A.9: Microprobe analyses of garnet and biotite from gold-mineralized hornfels from the Woolwonga Gold Mine 1 (Sample 93271). G = garnet, B = biotite.

93271	M1	M2	M3	M4	M5	M6	Ave.	K1	B6	B7	G-Ave.
SiO_2	45.86	46.12	47.51	47.20	48.68	46.60	47.00	64.44	34.3	33.34	36.77
TiO_2	0.14	0.26	0.29	0.24	0.22	0.21	0.23	<	1.22	1.29	<
Al_2O_3	34.21	33.52	33.21	32.64	31.01	33.45	33.01	17.8	18.32	18.68	20.94
FeO	1.57	2.00	1.83	2.00	2.16	1.68	1.87	0.32	23.37	23.87	17.24
MnO	<	<	<	<	<	<	<	<	0.51	0.52	23.34
MgO	0.14	0.49	0.5	0.63	1.19	0.35	0.55	<	7.49	7.92	0.19
CaO	<	<	<	<	<	<	<	<	<	<	2.76
Na_2O	<	<	<	<	<	<	<	<	<	<	0.04
K_2O	10.79	10.80	11.09	10.95	11.00	10.98	10.94	16.90	9.04	7.91	<
Total	92.72	93.20	94.42	93.65	94.25	93.25	93.58	99.47	94.27	93.55	101.28
n							6				5
Si p.f.u.	3.13	3.14	3.19	3.20	3.28	3.17	3.18				

Table A.10: Microprobe analyses from gold-mineralized hornfels from the Woolwonga Gold Mine 2 (Sampe 93271). B = biotite, G = garnet, K = K-feldspar, M = muscovite.

93384	1	2	3	4	5	6	7	8	Ave.	K1	K2
SiO_2	34.48	31.39	34.69	39.82	29.61	35.00	36.03	35.05	34.51	60.94	66.39
TiO_2	2.12	1.40	1.77	1.05	0.61	1.92	1.88	1.89	1.58	0.18	0.23
Al_2O_3	18.44	18.32	17.99	18.49	18.74	17.61	17.77	18.14	18.19	20.53	19.06
Fe2O3	20.21	24.68	22.54	16.71	28.6	18.97	19.06	20.55	21.42	0.16	0.20
MnO	0.21	0.16	0.25	0.23	0.26	0.17	0.09	0.14	0.19	<	<
MgO	8.75	8.81	8.51	9.43	7.58	9.90	9.98	9.58	9.07	0.14	<
CaO	<	<	<	<	<	<	<	<	<	<	<
Na_2O	0.03	0.03	0.07	0.03	0.04	0.04	0.07	0.05	0.05	0.47	0.53
K_2O	9.15	5.72	8.59	7.31	3.77	8.22	7.63	7.94	7.29	15.08	12.09
SO_3	333	<	236	<	<	<	<	<	71	181	<
F	8827	7184	9594	10640	7182	11979	15210	11026	10205	<	<
Cl	1878	1108	1105	1205	1715	1853	1137	1414	1427	58	138
Total	94.56	91.35	95.54	94.26	90.10	93.21	94.15	94.59	93.47	97.54	98.52
n									8		

Table A.11: K-spar (K) and biotite analyses from Au-vein selvage from the Enterprise Gold Mine (Sample 93384). K = K-feldspar.

93415	1	2	3	4	5	6	7	8	9	Ave.
SiO_2	51.85	52.34	51.68	52.08	51.50	51.63	50.62	51.61	52.31	51.74
TiO_2	<	<	<	0.10	<	<	0.10	<	0.18	0.04
Al_2O_3	3.15	2.74	2.61	2.43	3.17	2.48	4.55	3.27	4.67	3.23
FeO	13.54	11.33	12.69	11.84	11.91	12.36	11.94	10.95	11.58	12.02
MnO	0.56	0.50	0.54	0.69	0.47	0.49	0.86	0.60	0.62	0.59
MgO	13.95	15.77	15.41	15.70	15.09	15.35	14.89	15.85	15.90	15.32
CaO	12.56	12.21	11.44	11.83	12.23	12.03	11.51	11.66	11.68	11.91
Na_2O	0.14	0.24	<	0.19	0.24	0.15	0.41	0.26	0.39	0.22
K_2O	<	<	<	<	<	<	0.05	<	<	<
Total	95.75	95.3	94.37	94.84	94.62	94.48	95.29	94.38	97.34	95.15
n										9

Table A.12: Microprobe analyses of actinolite from a carbonaceous gold-vein selvage at the Woolwonga Gold Mine (Sample 93415).

SM11	1	2	3	4	5	6	7	8	9	10	ave.
SiO_2	36.41	36.86	36.02	36.41	36.12	36.32	36.25	36.23	36.85	37.13	36.46
TiO_2	2.48	1.96	2.11	1.69	1.33	2.22	2.39	2.44	2.19	2.28	2.11
Al_2O_3	17.27	17.20	17.01	16.99	17.25	16.62	16.59	16.58	16.66	16.02	16.82
FeO	15.14	15.54	15.17	15.14	15.33	15.45	15.79	15.71	15.44	15.38	15.41
MnO	0.67	0.66	0.71	0.64	0.60	0.63	0.71	0.71	0.64	0.55	0.65
MgO	11.79	12.24	12.00	12.25	11.81	12.08	12.15	12.12	12.54	12.82	12.18
CaO	0.05	0.05	0.05	<	<	<	<	<	<	<	<
Na_2O	0.04	0.06	0.04	0.05	0.06	0.05	0.06	0.05	0.06	0.08	0.06
K_2O	9.38	9.50	9.62	9.60	9.12	9.35	9.50	9.35	9.47	9.39	9.43
SO_3	181	<	149	<	443	238	<	<	358	<	137
F	4568	6510	4790	5219	8610	4864	5009	4982	5287	5887	5573
Cl	228	227	280	191	329	232	213	255	236	222	241
Total	93.72	94.70	93.23	93.32	92.62	93.27	93.36	93.70	94.49	94.26	93.67
n											10

Table A.13: Microprobe analyses of biotite from hornfels of the Phantom Pit at Cosmo Howley.

SM11 / 106558	G1	G2	G3	G4	G5	Ave.	B1	B2	B3	B4	B5	Ave.
SiO_2	37.21	37.49	36.99	37.31	37.36	37.27	33.98	35.46	35.27	34.86	35.11	34.94
TiO_2	0.27	0.19	<	<	<	<	1.35	1.35	1.38	1.36	1.41	1.37
Al_2O_3	20.61	20.88	20.98	20.85	21.13	20.89	16.93	16.83	16.77	16.89	16.94	16.87
FeO	10.06	11.63	11.60	13.43	10.58	11.46	21.08	21.13	19.87	21.05	20.56	20.74
MnO	21.45	19.95	20.13	19.11	20.29	20.19	0.08	<	0.09	<	<	<
MgO	0.47	0.62	0.54	0.85	0.66	0.63	10.25	10.35	10.48	10.62	10.16	10.37
CaO	8.67	8.21	8.60	7.16	9.26	8.38	<	<	<	<	<	<
Na_2O	0.01	<	0.01	<	<	<	0.12	0.21	0.17	0.21	0.18	0.18
K_2O	<	<	<	<	<	<	7.78	8.59	8.60	8.07	8.57	8.32
SO_3	<	0.05	<	<	<	<	154	<	<	<	224	63
F	<	<	<	<	<	<	6264	6409	6300	6853	6650	6616
Cl	<	51	<	<	<	<	525	568	636	664	607	607
Total	98.74	99.03	98.89	98.71	99.27	98.93	92.25	94.62	93.23	93.8	93.72	93.52
n						5						5

Table A.14: Microprobe analyses of garnet II from hornfels (2SM11) and biotite from carbonaceous slate (Sample 106558) at Cosmo Howley. G = garnet (2SM11), B = biotite (106558).

Appendix B

Abbreviations and Sample Catalog

In the text and tables in Part I, a set of self-consistent abbreviations for mineral names has been used according to the suggestion of Kretz (1983). These abbreviations and additional short-forms for rock types are listed below (Table B.1).

In the following the samples which are referred to in the text are compiled, referencing where they are stored, where more detailed descriptions can be obtained, and whether sample material is still available for future investigations.

Silicates and Carbonates			
Act	actinolite	Ilm	ilmenite
Alm	almandine	Kfs	potassium feldspar
Amph	amphibole	Mc	microcline
And	andalusite	Mnz	monazite
Ank	ankerite	Ms	muscovite
Ap	apatite	Qtz	quartz
Bt	biotite	Rt	rutile
Cal	calcite	Sd	siderite
Cld	chloritoid	Sco	scorodite
Crd	cordierite	Sps	spessartine
Chl	chlorite	Ttn	shene i.e. titanite
Drv	dravite	Tur	tourmaline
Dol	dolomite	Xe	xenotime
Grt	garnet	Zrn	zircon
Gr	graphite		

Oxides and Sulphides			
Apy	arsenopyrite	Lo	loellingite
Brt	barite	Mag	magnetite
Ccp	chalcopyrite	Py	pyrite, t-Py = tarnished pyrite
Gn	galena	Po	pyrrhotite
Hem	haematite	Sp	sphalerite

Rock Types, Veins	
bs	biotite schist
hf	hornfels
qhf	quartz-nodular hornfels or biotitic slate
QVx	quartz vein of generation x
UK	carbonaceous slate, Upper Koolpin Formation
WR	wallrock to a described structure

Table B.1: Abbreviations used in the sample descriptions in tables (eg., Kretz, 1983).

Samples referenced in Chapter 1					
Samples in RSES Catalog and Storage					
93190	93198	93227	93231	93233	93235
93245	93267	93271	93384	93415	93286
Samples consumed (Description upon request)					
AS218					

Table B.2: Storage of samples used in the study. More detailed descriptions of samples can be obtained from the author upon request.

Samples referenced in Chapter 2	
Samples consumed (Description upon request)	
AS10	AS11

Table B.3: Storage of samples used in the study. More detailed descriptions of samples can be obtained from the author upon request.

Samples referenced in Chapter 3	
Samples in RSES Catalog	
93097	93351

Table B.4: Storage of samples used in the study. More detailed descriptions of samples can be obtained from the author upon request.

Samples referenced in Chapter 4

Samples consumed (Matthäi, 1990, unpubl. comp. rep.)

AS20	AS21	AS22	AS23	AS24	AS25
AS26	AS27	AS28	AS29	AS30	AS31
AS32	AS33	AS34	AS35	AS76	AS77
AS78	AS79	AS80	AS81	AS82	AS83
AS84	AS85	AS86	AS87	AS88	AS89
AS90	AS91	AS92	AS93	AS94	AS95
AS96	AS97	AS98	AS99	AS100	AS 101
AS102	AS103	AS105	AS106	AS107	AS108
AS109	AS110	AS111	AS112	AS113	AS115
AS116	AS117	AS118	AS119	AS120	AS121
AS122	AS123	AS124	AS125	AS126	AS127
AS128	AS129	AS130	AS131	AS132	AS133
AS135	AS136	AS137	AS138	AS139	AS140
AS141	AS142	AS144	AS145	AS146	AS147
AS148	AS149	AS150	AS151	AS152	AS153
AS154	AS155	AS156	AS157	AS158	AS159
AS160	AS161	AS162	AS163	AS164	AS165
AS166	AS167	AS168	AS169	AS170	AS171
AS172	AS173	AS174	AS175	AS176	AS177
AS178	AS179	AS180	AS181	AS182	AS183
AS184	AS185	AS186	AS187	AS188	AS189
AS190	AS191	AS192	AS193	AS194	AS195
AS196	AS197	AS198	AS199	AS200	AS201
AS202	AS203	AS204	AS205	AS206	AS207
AS208	AS207	AS208	AS209	AS210	AS211
AS212	AS213	AS214	AS215	AS216	AS217
AS218	AS219	AS220	AS221	AS222	

Table B.5: Storage of samples used in the study.

Samples referenced in Chapter 5

Samples in RSES Catalog

93114	93126	93127	93133	93134/A
93134/1	93134/2	93134/3	93141	93149
93211	93212	93213	93215	93364
93367	93369	93376	93378	93380
93388/1	93388/2	93388/3	93388/4	93388/8
93424				

Samples described in CSIRO restr. rep. 111R

106551	06553	106554	106555	106555A
106557	106557A	106558	106559	106560
106561	06563	106561	106562	106562A
106562B	106563	106565	106567	106568
106571	106572	106574	106575	106576
106577				

7912-5013/1L2	7912-5013/1L3	7912-5013/2L2	8312-6025TOT	8312-6025/TL1
8312-6025/TL2	8312-6025/TL3	8312-6025/2L1	8312-6025/2L2	8312-6029TOT
8312-6029TL1	8312-6029/L2	8312-6029/L3	8312-6029/2L1	8312-6029/2L2
8312-60301L1	8312-603012	8312-60301L3	8312-60302L2	

Samples consumed, or not cataloged

DRCH1A	DRCH1B	DRCH2	DRCH4	DRCH4B
DRCH5				
SM77	2SM3/1A	2SM34/1A	3SM14	3SM16
3SM84	3SM94	3SM100/1	3SM100/2	3SM101

Table B.6: Storage of samples used in the study. More detailed descriptions of samples with the prefix SM and DR can be obtained from the author and R.W. Henley (Henly and Etheridge Geoscience Consultants, Deakin House, Suite 4, Geils Court 50, Deakin, ACT 2600), respectively.

Appendix C

Thermodynamic Calculations

In Part I results from speciation and reaction progress calculations are presented. In the following these calculations are described with more detail and the sources of the thermodynamic data are described. The newly fitted standard state properties for the $Au(HS)_2^-$ and $AuHS^0$ complex of gold are listed.

All thermodynamic calculations were carried out with a modified version of THERMODATA III (Turnbull and Wadsley, 1986) and the thermodynamic databases for minerals and aqueous species described in Shock and Helgeson (1988), Shock et al. (1989), and Holland and Powell (1990). The mineral database of Holland and Powell (1990) was modified according to the suggestions of Sverjensky et al. (1991). New solubility data for HCl, KCl, and NaCl of Sverjensky et al. (1991) were used. Additional thermodynamic data for the arsenic minerals and aqueous species were derived from Barton (1969) and Heinrich and Eadington (1986). The molar volumes of the arsenic minerals were taken from Sharp et al.(1985).

In the speciation, as well as in the reaction progress calculations a simple Debye-Hückel treatment for the charged species was employed using the solvent functions described in Helgeson and Kirkham (1974b) and McKenzie and Helgeson (1984). The unextended version of the Debye-Hückel equation was implemented following the suggestion by Sverjensky (1987) that the extended term parameter b becomes very small at supercritical pressures and temperatures.

C.1 Standard State Properties and Equation of State Parameters for Gold Complexes

The standard state properties for the aqueous gold complexes $AuHS^0$ and $Au(HS)_2^-$ were newly derived on the basis of the experimental data referenced in Table C.1.

On the basis of these basic data and initial guesses for the a_1 and a_3 parameters which were made using isocoulombic reaction theory (e.g., Ruaya, 1988), the fitting of the other parameters was commenced. The Born coefficient ω was estimated using the natural correlation with the entropy (Shock et al., 1989, Fig. 10, p. 2170). Because of this strong correlation a variation of this parameter seems hard to justify.

For the fitting of the a_1 to a_4, c_1, c_2 parameters the strategy suggested in Shock and Helgeson (1988) was followed broadly. Strong correlations between the c_1 and c_2, a_1 and a_2, and a_1 and a_3 parameters were obeyed to estimate missing parameters. With minor exeptions, these strong correlations can be observed for metal complexes

Species	ΔG_f^0	ΔH_f^0	$S^0_{P_r,T_r}$	a_1	a_2	a_3	a_4
$AuHS^0$		19439	70.277	-0.4167	2.8300	4.6356	-2.8900
$Au(HS)_2^-$		5584	78.471	-0.0630	15.7763	-7.9114	3.6930
$AuCl_2^-$	-36790	-47296	45.000	0.0885	15.4203	-3.7394	-3.4165

Species	c_1	c_2	ω	Reference
$AuHS^0$	-95.1758	-5.8800	-63000	(1)
$Au(HS)_2^-$	-48.4764	-2.1110	44100	(2)
$AuCl_2^-$	20.6850	-0.9976	93973	(3)

Table C.1: Newly derived and existing fitting parameters for gold complexes to the modified equation HKF equation of state for aqueous species (Tanger and Helgeson, 1988). Units: ΔG_f^0 $(cal\,mol^{-1})$, ΔH_f^0 $(cal\,mol^{-1})$, $S^0_{P_r,T_r}$ $(cal\,mol^{-1}\,K^{-1})$, $a_1(10^1\,cal\,mol^{-1}\,bar^{-1}$, $a_2(10^{-2}\,cal/mol)$, $a_3(10^0\,cal\,K\,mol^{-1}\,bar^{-1}$, $a_4(10^{-4}\,cal\,K\,mol^{-1})$, $c_1(10^0\,cal\,mol^{-1}\,K^{-1}$, $c_2(10^{-4}\,cal\,K\,mol^{-1}$, $\omega_{10^{-5}P_r,T_r}(cal\,mol^{-1})$. References: (1) Seward, 1990, Renders and Seward, 1989, Hayashi and Ohmoto, 1991., Benning and Seward, 1993, Gibert et al., 1993. (2) Renders and Seward, 1989, Seward, 1974, Seward, 1990, Schenberger and Barnes, 1989. (3) Zotov et al., 1991.

of similar charge by examination of the entries in the SUPCRT93 database. In a further step, published solubility data were compared to calculated values obtained with the preliminary fits. To obtain a feel for how changes in the fitting-parameters affect the shape of the solubility curves, a sensitivity analysis was conducted by systematic variation of individual parameters, but still obeying the identified interdependencies. In general, appreciable variation of the fitting parameters does not result in strong changes of the predicted solubilities. The difficulty of fitting consisted in satisfying constraints from different experiments simultaneously. This was done more or less in a visual "least squares" type approach. Importantly, only the pressure sensitivity (molar volume data) of the $AuCl_2^-$ complex appears to be well constrained by the experimental data presented in Zotov et al. (1991). For the other complexes the fits still rely quite heavily on the predictive capabilities of the revised equation of state (Tanger and Helgeson, 1988, Shock and Helgeson, 1988, Shock et al., 1989).

C.2 Reaction Progress Calculations

The efficiency of gold precipitation mechanisms was investigated using the program CHEMIX (a subroutine of Thermodata). As an example of the composition of the fluid prior to titration with equal amounts of carbon-dioxide and methane, the following speciation obtained at 550°C, 200 MPa in the presence of the assemblage Mag + Po + Kfs + Ab + Mu (1 molal Cl) is shown in Table C.2. This listing is used also to show which aqueous species were included in the other speciation calculations presented in this thesis, except for the arsenic minerals.

Aqueous Species	Moles	Grams	Molality	Activity
H_2O	45.26	815.4	55.51	0.9703
H_2	0.6511E-01	0.1313	0.7985E-01	0.7985E-01
H_2S	0.2655	9.047	0.3256	0.3256
HS^-	0.6013E-04	0.1989E-02	0.7375E-04	0.7427E-04
SO_2	0.7802E-06	0.4998E-04	0.9568E-06	0.9568E-06
SO_4^{-2}	0.6268E-12	0.6021E-10	0.7688E-12	0.7581E-12
HSO_4^-	0.9752E-09	0.9466E-07	0.1196E-08	0.1204E-08
Na^+	0.1089	2.503	0.1335	0.1371
OH^-	0.1583E-05	0.2692E-04	0.1941E-05	0.1955E-05
H^+	0.8437E-05	0.8500E-05	0.1035E-04	0.1042E-04
Cl^-	0.1236	4.381	0.1516	0.1524
HCl	0.1086E-02	0.3961E-01	0.1332E-02	0.1332E-02
NaCl	0.6854	40.06	0.8406	0.8406
$FeCl_2$	0.7020E-01	8.898	0.8609E-01	0.8609E-01
$AuCl_2^-$	0.4991E-06	0.1337E-03	0.6121E-06	0.6121E-06
$Au(HS)_2^-$	0.5247E-07	0.1381E-04	0.6435E-07	0.6435E-07
$AuHS^0$	0.2965E-06	0.6820E-04	0.3636E-06	0.3636E-06
K^+	0.1475E-01	0.5767	0.1809E-01	0.1818E-01
KCl	0.4960E-01	3.698	0.6083E-01	0.6083E-01
$FeCl^+$	0.4010E-05	0.3661E-03	0.4918E-05	0.4918E-05
KOH	0.3147E-05	0.1766E-03	0.3859E-05	0.3859E-05

Table C.2: Composition of gold-bearing fluid at 550°C, 200 MPa. This speciation was obtained for the buffer assemblage Mag + Po + Kfs + Ab + Mu (1 molal Cl).

Appendix D

Source Code of the Monte Carlo Method

The following source code is written is ANSI-C and should compile on any current C-compiler under the UNIX operating system. This does not apply to the alternative, parallel C*-code in the function `parallel_walk.cs` which is specific to the CONNECTION MACHINE CM5 from Think Machines Corperation. The parallel code also draws on non-standard library routines for the generation of parallel lookup tables. These library routines are hardware-specific and unique to the 1993-setup of the CM5 at the Australian National University.

The input matrix for the Monte Carlo program can be any 256-color dump from an image. The interpretation of the color values is specified on the help screen. The input file should be formatted as text which contains only the matrix entries. Each row has to be terminated by a line-break and the file should not contain any blank lines at the end, because these upset the automatic determination of the matrix size.

If this code or modifications thereof are used for numerical modelling the results of which are published, reference should be made in the publication to the originator. This applies, especially to incorporated memory allocation functions which have been published by Press et al., 1988 (Numerical Recipes in C, Cambridge University Press).

```
/*-----------------------------------------------------*/
/* mc.c  : MAIN 2d RANDOM WALK FLOW SIMULATION PROGRAM */
/*-----------------------------------------------------*/

#include "mtcarlo.h"
#include "m_intro.c"      /* introduces the program and help          */
#include "slurp.c"        /* reads image rainbow-color coded text file */
#include "convert.c"      /* converts color ID's to permeabilities    */
#include "sm_util.c"      /* memory allocation functions            */
#include "nrutil.c"       /* memory allocation functions            */
#include "read_b.c"       /* reads self-made boundary conditions      */
#include "bound.c"        /* creates simple boundary conditions       */
#include "walk_addrans.c" /* Mt Carlo simulation                */
#include "write_perm.c"   /* writes permeability matrix to *.perm     */

  void
main ()
  {
    extern int m_mtrx, n_mtrx;          /* grid dimensions             */
    extern int m_scale, n_scale;        /* physical model dimensions   */
    unsigned long **picture;            /* storage of permeabilities   */
    int c;                              /* Yes/No input                */
    enum { N, Y } own;                  /* Yes/No query result         */
    double **P_fluid; /* Dirichlet boundary condit.          */

    char pict_file[MAXSTR], perm_file[MAXSTR], result_file[MAXSTR],
         bound_file[MAXSTR], dummy[LARGESTR];

    /* EXTERNAL FUNCTION PROTOTYPES */
    unsigned long **slurp_picture ( char *infilename );
    unsigned long **pict_to_perm ( unsigned long **data );
    void free_ulmatrix(unsigned long **m, int nrl, int nrh, int ncl);
    void free_dmatrix(double **m, int nrl, int nrh, int ncl);
    void m_intro ();
    void write_perm( unsigned long **perm, char outfile[MAXSTR] );
    double **read_bound ( char *bound_file );
    double **boundary ( char *bound_file, unsigned long **perm );
    void walk ( unsigned long **perm, double **bound, char outfile[MAXSTR] );


    /* DATA AQUISITION AND CONVERSION */
    m_intro ();
    gets( dummy );
    printf("\nMain: Please specify matrix-image [text] file : ");
    gets ( pict_file );
    picture = slurp_picture ( pict_file );
    pict_to_perm ( picture );

    /* SOME CONTROL AND FILE KEEPING */
    strncpy ( result_file, pict_file, NAME );
    strcat ( result_file, ".res");  /* append extension RES */
    strncpy ( bound_file, pict_file, NAME );
    strcat ( bound_file, ".bnd");   /* append extension BND */
    strncpy ( perm_file, pict_file, NAME );
    strcat ( perm_file, ".perm");   /* append extension BND */

    printf("\nMain: Size of image: M = %d; N = %d\n", m_mtrx, n_mtrx );
```

```
  printf("\nMain: Please enter depth extend of model [m] : ");
  scanf("%d", &m_scale );
  n_scale = m_scale;
  gets( dummy );

  /* OUTPUT PERMEABILITY MATRIX FOR PLOTTING */
  write_perm( picture, perm_file );

  /* APPLICATION OF BOUNDARY CONDITIONS */
  question : { printf ("\nDo already have boundary conditions [Y,N] : ");
               c = getchar();
               if ( c == 89 ) own = Y;
               if ( c == 78 ) own = N;
               if ( c != 89 && c != 78 )
                  {
                    c = getchar();
                    goto question;
                  }
             }
  if ( own )
    {
      gets ( dummy );
      printf("\nMain: Please specify boundary condition file [text] : ");
      gets ( bound_file );
      P_fluid = read_bound ( bound_file );
    } else {
             P_fluid = boundary ( bound_file, picture );
             printf("\nMain: A linear fluid pressure gradient has been\n");
             printf("applied to the vertical model boundaries\n");
           }

  /* MT CARLO SIMULATION */
  walk ( picture, P_fluid, result_file );

  /* MEMORY LIBERATION */
  free_ulmatrix( picture, 1, m_mtrx, 1);
  free_dmatrix( P_fluid, 1, m_mtrx, 1);

} /* end main */
```

```
/*-------------------------------------------------------------*\
|                                                               |
| mtcarlo.h :  GLOBALS and DEFINITIONS for Monte Carlo Modelling |
|                                                               |
\*-------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "nr.h"
#include "nrutil.h"
#include <ctype.h>
#include <string.h>

#define MAXSTR    75        /* max length of DOS filename              */
#define MAXERR    90        /* max length of error message             */
#define NAME      25        /*  "     "   of filename without extension */
#define MAXSIZE   100001    /* max NxM size of image matrix            */
#define LARGESTR  100       /* max stringlength of screen input        */
#define IN  1               /* switches for word counting              */
#define OUT 0
#define YEP 1
#define NOP 0


int m_mtrx, n_mtrx;         /* size of the permeability matrix         */
int m_scale, n_scale;       /* physical size of the model              */
```

```
/*-------------------------------------------------------------*/
/* sm_util.c : modified functions from Numerical Recipes in C */
/*    from Press et al., 1988, Cambridge Univ. Press */
/*             modifications for ANSI-standard C and some     */
/*             new functions by Stephan K. Matthai 3/90       */
/*-------------------------------------------------------------*/

 #include <malloc.h>
 #include <stdio.h>
 #include <stdlib.h>

 #define MAXERR 100

void nrerror(char error_text[MAXERR])
{
  fprintf(stderr,"nerror at run-time...\n");
  fprintf(stderr,"%s\n",error_text);
  fprintf(stderr,"...now exiting to system...\n");
  exit( 1 );
}

float *vector(int nl,int nh)
  {
    float *v;
    v=(float *)malloc((unsigned) (nh-nl+1)*sizeof(float));
    if (!v) nrerror("allocation failure in vector()");
    return v-nl;
  }

int *ivector(int nl, int nh)
  {
    int *v;
    v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
    if (!v) nrerror("allocation failure in ivector()");
    return v-nl;
  }

double *dvector(int nl,int nh)
  {
    double *v;
    v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
    if (!v) nrerror("allocation failure in dvector()");
    return v-nl;
  }

void free_vector(float *v, int nl)  /* nh is never used */
  {
    free((char*) (v+nl));
  }

void free_ivector(int *v, int nl)
  {
    free((char*) (v+nl));
  }

void free_dvector(double *v, int nl)
  {
```

```
    free((char*) (v+nl));
  }

float **matrix(int nrl,int nrh,int ncl,int nch)
{
  int i;
  float **m;
  m=(float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float*));
  if (!m) nrerror("allocation failure 1 in matrix()");
  m -= nrl;
  for(i=nrl;i<=nrh;i++)
   {
     m[i]=(float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));
     if (!m[i]) nrerror("allocation failure 2 in matrix()");
     m[i] -= ncl;
   }
return m;
}


double **dmatrix(int nrl,int nrh,int ncl,int nch)
{
  int i;
  double **m;
  m=(double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
  if (!m) nrerror("allocation failure 1 in dmatrix()");
  m -= nrl;
  for(i=nrl;i<=nrh;i++)
   {
     m[i]=(double *) malloc((unsigned) (nch-ncl+1)*sizeof(double));
     if (!m[i]) nrerror("allocation failure 2 in dmatrix()");
     m[i] -= ncl;
   }
return m;
}

int **imatrix(int nrl,int nrh,int ncl,int nch)
{
  int i,**m;
  m=(int **)malloc((unsigned) (nrh-nrl+1)*sizeof(int*));
  if (!m) nrerror("allocation failure 1 in imatrix()");
  m -= nrl;
  for(i=nrl;i<=nrh;i++)
    {
      m[i]=(int *)malloc((unsigned) (nch-ncl+1)*sizeof(int));
      if (!m[i]) nrerror("allocation failure 2 in imatrix()");
      m[i] -= ncl;
    }
return m;
}

unsigned long **ulmatrix(int nrl,int nrh,int ncl,int nch)
{
  int i;
  unsigned long **m;
  m=(unsigned long **)malloc((unsigned) (nrh-nrl+1)*sizeof(unsigned
long*));
  if (!m) nrerror("allocation failure 1 in ulmatrix()");
```

```
  m -= nrl;
  for(i=nrl;i<=nrh;i++)
    {
     m[i]=(unsigned long *)malloc((unsigned) (nch-ncl+1)*sizeof(unsigned long));
      if (!m[i]) nrerror("allocation failure 2 in ulmatrix()");
      m[i] -= ncl;
    }
return m;
}

void free_matrix(float **m,int nrl,int nrh,int ncl)
{
  int i;
  for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
  free((char*) (m+nrl));
}

void free_dmatrix(double **m,int nrl,int nrh,int ncl)
{
  int i;
  for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
  free((char*) (m+nrl));
}

void free_imatrix(int **m,int nrl,int nrh,int ncl)
{
  int i;
  for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
  free((char*) (m+nrl));
}

void free_ulmatrix(unsigned long **m,int nrl,int nrh,int ncl)
{
  int i;
  for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
  free((char*) (m+nrl));
}

/* function allocates memory for a three-dimensional array */
/* of the range: [nrl,nrh][ncl,nch][vcl,vch]               */

double ***d3matrix(int nrl,int nrh,int ncl,int nch,int vcl,int vch)
 {
   int i, j;
   double ***m;

   /* Allocate pointers to rows */
   m=(double ***) malloc((unsigned) (nrh-nrl+1)*sizeof(double**));
   if (!m) nrerror("allocation failure 1 in 3matrix()");
   m -= nrl;

   /* Allocate rows and set pointers to them */
   for(i=nrl;i<=nrh;i++)
     {
       m[i]=(double **) malloc((unsigned) (nch-ncl+1)*sizeof(double*));
       if (!m[i]) nrerror("allocation failure 2 in 3matrix()");
       m[i] -= ncl;
     }
```

```
   /* Allocate row element vectors and set pointers to them */
   for(i=nrl;i<=nrh;i++)
     {
       for (j=ncl;j<=nch;j++)
         {
           m[i][j]=(double *) malloc((unsigned) (vch-vcl+1)*sizeof(double));
           if (!m[i][j]) nrerror("allocation failure 3 in 3matrix()");
           m[i][j] -= vcl;
         }
     }
   return m;
 }

/* function free's a three dimensional memory array of the range : */
/* [nrl,nrh][ncl,nch][vcl] */

  void
free_d3matrix(double ***m, int nrl,int nrh,int ncl,int nch,int vcl)
{
 int i, j;

    for(i=nrh;i>=nrl;i--)
      {
        for (j=nch;j>=ncl;j--) free((char*) (m[i][j]+vcl));
      }
    for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
 free((char*) (m+nrl));
}

float *vector(int nl,int nh)
  {
    float *v;
    v=(float *)malloc((unsigned) (nh-nl+1)*sizeof(float));
    if (!v) nrerror("allocation failure in vector()");
    return v-nl;
  }

int *ivector(int nl, int nh)
  {
    int *v;
    v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
    if (!v) nrerror("allocation failure in ivector()");
    return v-nl;
  }

double *dvector(int nl,int nh)
  {
    double *v;
    v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
    if (!v) nrerror("allocation failure in dvector()");
    return v-nl;
  }

void free_vector(float *v, int nl)  /* nh is never used */
  {
    free((char*) (v+nl));
  }
```

```
void free_ivector(int *v, int nl)
  {
    free((char*) (v+nl));
  }

void free_dvector(double *v, int nl)
  {
    free((char*) (v+nl));
  }

/*-------------------------------------------------------*/
/* zero_double.c : Zeroes members of arrays of doubles */
/*-------------------------------------------------------*/

  void
zero_double( double **data, int m, int n )
 {
   int i, j;

   for( i=1;i<=m;i++ )
     for( j=1;j<=n;j++ ) data[i][j] = 0.0;

 } /* end zero_double */


/*-------------------------------------------------------*/
/* zero_ulong.c : Zeroes members of arrays of unsigned long */
/*-------------------------------------------------------*/

  void
zero_ulong( unsigned long **data, int m, int n )
 {
   int i, j;

   for( i=1;i<=m;i++ )
     for( j=1;j<=n;j++ ) data[i][j] = 0;

 } /* end zero_ulong */
```

```
 /*-------------------------------------------------------------------*\
 |                                                                     |
 | m_intro.c  : introduction to perm matrix and boundary generation    |
 |                                                                     |
 \*-------------------------------------------------------------------*/

/* #include "mtcarlo.h" */

void m_intro ()
  {
    int c;
    enum { N, Y } help;

printf("\n\n\n\n* ======================================================
*\n\n");
printf("*  *    MONTE CARLO 2d RANDOM WALK FLOW SIMULATION    *  *\n\n");
printf("*  *      using 256 color-decoding of text files      *  *\n");
printf("*  *          Vs. 1.8 by S.K. Matthai 8/93            *  *\n\n");
printf("* ====================================================== *\n\n");
printf("\n\n\n\n\n\n\n\n\n\n");

    question : { printf ("Type [Y,N] for help (color information etc.): ");
                 c = getchar();
                 if ( c == 89 ) help = Y;
                 if ( c == 78 ) help = N;
                 if ( c != 89 && c != 78 )
                    {
                      c = getchar();
                      goto question;
                    }
               }

    if ( help )
      {
printf("\nColor -> permeability conversion code (Rainbow scheme) :\n\n");
printf("        100,000 = highest      K = green\n");
printf("         10,000 = sec.highest  K = yellow\n");
printf("          1,000 = high         K = red\n");
printf("            100 = intermediate K = white\n");
printf("             10 = low          K = blue\n");
printf("              1 = lowest       K = black\n\n");
printf("The program will exit if other colors are used !\n");
printf("\nThe size of the image-matrix is detected automatically\n");
printf("\nUse a fluid pressure range which is about the same as-");
printf("\nor smaller than the value at the boundary on the");
printf("\nlow-pressure side. This improves the sampling");
printf("\nstatistics i.e. solution scatter.\n\n");
      }
  }
```

```
/*-------------------------------------------------------------------------*/
/* slurp.c : picture matrix aquisition routine which checks the size of   */
/*           a matrix presented as [text - integer] input file and reads  */
/*           it in an array. It takes the  input file name and returns a  */
/*           pointer to the array.                          */
/*           by Stephan K. Matthai 8/92                */
/*        NOTE: input file should start with the first data point a[1][1] */
/*-------------------------------------------------------------------------*/

unsigned long **slurp_picture ( char *infilename )
  {
     int max_m, max_n, i,j, state, m_elements;
     unsigned long **data;
     char dummy[MAXSTR];
     FILE *fp;
     extern int n_mtrx, m_mtrx;     /* global matrix size parameters */
     unsigned long **ulmatrix(int nrl, int nrh, int ncl, int nch);
     void nrerror(char error_text[MAXSTR]);

      /* CHECKING THE SIZE OF MATRIX */
      if ((fp = fopen ( infilename,"r")) == NULL)
         nrerror("Data file not available !");
      while (!feof(fp))
        {
          state = OUT;
          n_mtrx = m_mtrx = i = j = m_elements = 0;
          while ((i = fgetc( fp )) != EOF )
            {
               if ( isalpha(i) )  /* test file for text */
                 {
                   printf("File contains text !?!\n");
                   printf("Unable to continue reading\n");
                   exit ( 1 );
                 }
               if (i == '\n') ++m_mtrx;
               if (i == ' ' || i == '\n' || i == '\t')
                   state = OUT;
               else if (state == OUT)
                 {
                   state = IN;
                   ++m_elements;
                 }
            }
        }
      n_mtrx = m_elements / m_mtrx;
      rewind ( fp );
      /* READING THE DATA INTO MEMORY */
      data = ulmatrix ( 1, m_mtrx, 1, n_mtrx );
      for (i = 1; i <= m_mtrx; i++)
         for (j = 1; j <= n_mtrx; j++) fscanf( fp,"%d", &data[i][j] );
    fclose( fp );
  return ( data );
 }
```

```
/*-------------------------------------------------------------------*/
/* bound.c :  writes a textfile with Dirichlet boundary conditions. */
/*            The function applies a vertical P-gradient which can  */
/*            be modified manually(by grepping the node coordinate).*/
/*-------------------------------------------------------------------*/

double **boundary( char bound_file[MAXSTR], unsigned long **perm )
  {
     extern int m_mtrx, n_mtrx;
     int i,j;
     double lower, upper, lower_l, lower_r, local, **b_cond;
     double pl, pr, r_K_total = 0.0, l_K_total = 0.0;
     FILE *fp;
     double **dmatrix(int nrl, int nrh, int ncl, int nch);

     printf("\nBoundary: Enter upper and lower fluid pressure : ");
     scanf("%lf %lf", &upper, &lower ); /* upper refers to pressure */

     b_cond = dmatrix ( 1, m_mtrx, 1, n_mtrx );

     /* getting permeability step sum on the left vertical boundary */
     /* 1/K because high K = high number.                           */
     l_K_total += ( (1/(double)perm[1][1]) * 0.5);
     l_K_total += ( (1/(double)perm[m_mtrx][1]) * 0.5);
     for( i=2; i<m_mtrx; i++ ) l_K_total += (1/(double)perm[i][1]);

     /* getting permeability step sum on the right vertical boundary */
     r_K_total += ( (1/(double)perm[1][n_mtrx]) * 0.5);
     r_K_total += ( (1/(double)perm[m_mtrx][n_mtrx]) * 0.5);
     for( i=2; i<m_mtrx; i++ ) r_K_total += (1/(double)perm[i][n_mtrx]);

        if ((fp = fopen ( bound_file,"wt")) == NULL)
            nrerror("Data file could not be opened !");
        fprintf( fp,"Computer generated boundary file\n");
        fprintf( fp,"%d  %d\n", m_mtrx, n_mtrx );
        for (j = 1; j <= n_mtrx; j++)    /* prints lower boundary */
               {
                 fprintf( fp,"1  %d %le\n", j, lower );
                 b_cond[1][j] = lower;
               }
        for (j = 1; j <= n_mtrx; j++)   /* prints upper boundary */
               {
                 fprintf( fp,"%d  %d %le\n", m_mtrx, j, upper );
                 b_cond[m_mtrx][j] = upper;
               }

        /* defining total distributed pressure/potential drop */
        local = upper - lower;
        lower_l = lower_r = lower;

        /* writing the boundaries */
        for (i = 2; i < m_mtrx; i++)
               {
                 /* left vertical boundary */
 pl = (1/(double)perm[i-1][1]) * 0.5 + (1/(double)perm[i][1]) * 0.5;
                 lower_l += ( local * pl / l_K_total );
                 fprintf( fp,"%d  1  %le\n", i, lower_l );
```

```
                b_cond[i][1] = lower_l;

                /* right vertical boundary */
pr = (1/(double)perm[i-1][n_mtrx]) * 0.5 + (1/(double)perm[i][n_mtrx]) * 0.5;
                lower_r += ( local * pr / r_K_total );
                fprintf( fp,"%d  %d  %le\n", i, n_mtrx, lower_r );
                b_cond[i][n_mtrx] = lower_r;
              }

    printf("\nBoundary: [%s] has been written successfully\n", bound_file );
    fclose(fp);
   return ( b_cond );
 }
```

```
/*-------------------------------------------------------------*\
|                                                               |
| read_b.c : reads Dirichlet bound. cond. from file into array  |
|                                                               |
\*-------------------------------------------------------------*/

 double
**read_bound ( char *bound_file )
 {
    extern m_mtrx, n_mtrx;
    int i, j;
    unsigned long boundary, lz;
    char dummy[MAXSTR];
    double **P_fluid, value;
    double **dmatrix(int nrl, int nrh, int ncl, int nch);
    void nrerror(char error_text[MAXSTR]);
    FILE *b_in;

   if ((b_in = fopen( bound_file,"rt")) == NULL )
     nrerror("Data file not found !\n");
   P_fluid = dmatrix( 1, m_mtrx, 1, n_mtrx);
   while ( !feof(b_in) )
     {
       fgets (dummy, MAXSTR, b_in);
       fgets (dummy, MAXSTR, b_in);
       boundary = (2 * m_mtrx) + (2 * n_mtrx) - 3;
       for (lz = 1; lz <= boundary; lz++)
         {
            fscanf (b_in, "%d %d %lf", &i, &j, &value);
            P_fluid[i][j] = value;
         }
     } /* END WHILE LOOP */
  fclose(b_in);    /* Data are aquired and file is no longer needed */
  printf ("\n[%s] has been read successfully\n\n", bound_file );
  return ( P_fluid );
 }
```

```
/*-------------------------------------------------------------*\
|                                                               |
| convert.c :  Function converts 256 colors (rainbow scheme)    |
|                to permeability values ranging from 1 to 100k. |
|                                                               |
|                Convention:100,000 = highest1     K = green    |
|                           10,000 = highest       K = yellow   |
|                            1,000 = high          K = red      |
|                              100 = intermediate  K = white    |
|                               10 = low           K = blue     |
|                                1 = lowest        K = black    |
|                                                               |
\*-------------------------------------------------------------*/

#define HIGHEST1     100000
#define HIGHEST       10000
#define HIGH           1000
#define INTERMEDIATE    100
#define LOW              10
#define LOWEST            1
#define WHITE     0
#define BLACK   255

unsigned long **pict_to_perm ( unsigned long **data )
   {
      int i, j;
      extern int m_mtrx, n_mtrx;
      void free_ulmatrix( unsigned long **m, int i, int j, int c );

      for (i = 1; i <= m_mtrx; i++)
        {
          for (j = 1; j <= n_mtrx; j++)
            {
              /* COLOR - PERMEABILITY CONVERSION */
              if ( data[i][j] == BLACK ) data[i][j]    = LOWEST;
              else if ( data[i][j] == WHITE ) data[i][j] = INTERMEDIATE;
              else if ( data[i][j] <=  65 ) data[i][j] = LOW;     /* blue   */
              else if ( data[i][j] <= 190 ) data[i][j] = HIGHEST1;/* green  */
              else if ( data[i][j] <= 225 ) data[i][j] = HIGHEST; /* yellow */
              else if ( data[i][j] <= 254 ) data[i][j] = HIGH;    /* red    */
              else
                {
                  printf("\nError during color conversion. Use only the");
                  printf("\ncolors specified in the help menu !");
                  printf("\nNow exciting...\n\n\n");
                  free_ulmatrix( data, 1, m_mtrx, 1 );
                  exit( -1 );
                }
            }
        }
      return ( data );
   }
```

```
/*-------------------------------------------------------------*/
/* write_perm.c : writes permeability matrix in specific format:*/
/*  see Spyglass Transform 3.1 ASCII-special format, manpage 62 */
/*-------------------------------------------------------------*/

void write_perm ( unsigned long **perm, char outfile[MAXSTR] )
  {
    /* WALK PARAMETERS */
    int i, l, lz;
    FILE *r_out;      /* output file */

    /* LOGICAL AND RESULT PARAMETERS */
    double *x_scale, *z_scale;

    /* EXTERN VARIABLES AND FUNCTIONS */
    extern int m_mtrx, n_mtrx;
    extern int m_scale, n_scale;
    void nrerror(char error_text[MAXERR]);

    /* CREATING MODEL SCALES */
    z_scale = dvector( 1, m_mtrx );
    x_scale = dvector( 1, n_mtrx );
    for( i=1; i<=m_mtrx; i++ ) z_scale[i] = i * m_scale/m_mtrx;
    for( i=1; i<=n_mtrx; i++ ) x_scale[i] = i * n_scale/n_mtrx;

    /* RESULTS ARE PRINTED TO FILE */
    if ((r_out = fopen( outfile,"wt")) == NULL )
      {
        nrerror("Cannot open output file !\n");
        exit( 1 );
      }
    fprintf (r_out,"%d %d\n", m_mtrx, n_mtrx );
    fprintf (r_out,"1  100000\n");        /* permeability range   */
    for (lz = 1; lz <= m_mtrx; lz++)      /* inserting row labels */
      fprintf (r_out, "%le ", z_scale[lz] );
    fprintf (r_out, "\n");
    for (lz = 1; lz <= n_mtrx; lz++)   /* inserting column labels */
      fprintf (r_out, "%le ", x_scale[lz] );
    fprintf (r_out, "\n");

    for (l = 1; l <= m_mtrx; l++)  /* writing permeability matrix */
      {
        for (lz = 1; lz <= n_mtrx; lz++)
         {
            fprintf (r_out, "%ld ", perm[l][lz] );
          }
        fprintf (r_out,"\n");
      }
    fprintf (r_out, "\n\n");
    fclose( r_out );

   /* END OF FUNCTION */
  }
```

```
/*-------------------------------------------------------------------*/
/* walk_addrans.c : MtCarlo random walk simulation of Pf-distribution. */
/*                  Version 1.9 1/8/93, using 'addrans' random numbers */
/* by S.K. Matthai for ANSI-C on a serial machine. With this code PhD */
/*             results were obtained.                  */
/*-------------------------------------------------------------------*/

#include "zero_double.c"
#include "zero_ulong.c"
#include "zero_int.c"

#define LARGE_MTRX 1000   /* Number of random walks for p[x][y] */
#define ADDRAN_SIZE 255   /* maxarray for random numbers      */
#define N_WALKS    1000   /* number of random walks per node    */

void intro()
  {

printf("\n*********************************************************\n\n");
    printf("           MONTE CARLO SIMULATION OF FLUID FLOW\n");
    printf("    (S.K.Matthai 8/93; Time-consuming on small machines !!!)\n\n");
printf("*********************************************************\n\n");
  }

 void
walk ( unsigned long **perm, double **bound, char outfile[MAXSTR] )
  {
    /* WALK PARAMETERS */
    register x, y, z, l;                /* fast loop var., array index */
    int i, j, lz, boundary;             /* move operators              */
    unsigned int p1, p2, p3, p4, p,     /* permeability matrix values  */
                 max_w, min_w;          /* walk performance indicators */

    /* DATA ARRAYS */
    double **val_counter;               /* b-values and result         */
    unsigned long **walk_counter;       /* counts walks for each node  */
    int **track;                        /* activates nodes along walk  */
    FILE *r_out;                        /* output file */

    /* LOGICAL AND RESULT PARAMETERS */
    enum { YES, NO } limit;
    short l_up, l_right, l_down, l_left;
    unsigned int right, down;
    unsigned int ran;
    double *x_scale, *z_scale;

    /* RANDOM GENERATOR PARAMETERS */
    unsigned lb = 1, xran[1];
    int nran = 1;

    /* EXTERN VARIABLES AND FUNCTIONS */
    extern int m_mtrx, n_mtrx;
    extern int m_scale, n_scale;
    void intro();
    void nrerror(char error_text[MAXERR]);
    double **dmatrix(int nrl, int nrh, int ncl, int nch);
    unsigned long **ulmatrix(int nrl,int nrh,int ncl,int nch);
```

```
void free_dmatrix(double **m,int nrl,int nrh,int ncl);
void free_ulmatrix(unsigned long **m,int nrl,int nrh,int ncl);
void zero_double( double **data, int m, int n );
void zero_ulong( unsigned long **data, int m, int n );
void zero_int( int **data, int m, int n );


/* CREATING MODEL SCALES */
z_scale = dvector( 1, m_mtrx );
x_scale = dvector( 1, n_mtrx );
for( i=1; i<=m_mtrx; i++ ) z_scale[i] = i * m_scale/m_mtrx;
for( i=1; i<=n_mtrx; i++ ) x_scale[i] = i * n_scale/n_mtrx;

/* START OF FUNCTION */
intro();           /* Prints introduction on the screen  */
i_init_addrans_(); /* initialize random number generator */

/* SETTING UP REQUIRED ARRAYS */
val_counter  = dmatrix (1,m_mtrx,1,n_mtrx);
walk_counter = ulmatrix (1,m_mtrx,1,n_mtrx);
track        = imatrix (1,m_mtrx,1,n_mtrx);

/* SETTING COUNTERS TO ZERO */
zero_double( val_counter, m_mtrx, n_mtrx );
zero_ulong( walk_counter, m_mtrx, n_mtrx );

/* RANDOM WALK FUNCTION */
for (lz = 2; lz < m_mtrx; lz++)
  {
    for (l = 2; l < n_mtrx; l++)
      {
       /* START OF WALKING */
       for ( z = 1; z <= N_WALKS; z++ )
       {
         zero_int( track, m_mtrx, n_mtrx ); /* no walk-history */
         x = lz;          /* Starting point of walk */
         y = l;
         track[x][y]++;  /* activates point along the track */
         l_up = l_right = l_down = l_left = NOP;
         limit = NO;
       /* START INDIVIDUAL WALK */
       while ( limit != YES )
         {
        /* with log-type K-scaling (Kij+Kmn)/2 approx. by Kmn */
        /* in neighborhood only, saves mighty runtime         */
        p1 = perm[(x-1)][y];  /* up */
        p2 = perm[x][(y+1)];  /* right */
        p3 = perm[(x+1)][y];  /* down */
        p4 = perm[x][(y-1)];  /* left */
        /* there's no way back to previous position */
        if ( l_up == YEP ) p3 = 0;
        if ( l_right == YEP ) p4 = 0;
        if ( l_down == YEP ) p1 = 0;
        if ( l_left == YEP ) p2 = 0;
        right = p1 + p2;
        down = right + p3;
        p  = down + p4;
        l_up = l_right = l_down = l_left = NOP;
```

```
if ( p == 0 )   /* WALK SAFETY */
  {
    printf ("\nfunction walk: Questionable coordinate %d %d\n",
            x, y );
    printf ("after %d random walks !\n", z );
    exit ( 1 );
  }
/* NOT PORTABLE */
u_addrans_( xran, &nran, &lb, &p );       /* p is upper bound */
ran = xran[0];            /* makes the required random number */
/* PROBABILISTIC DECISION */
if ( ran <= p1 )
  {
    x--;           /* Walks up */
    l_up = YEP;
    track[x][y]++; /* Activates point along the track */
  } else if ( ran > p1 )
     {
       if ( ran <= right )
         {
           y++;     /* Walks right */
           l_right = YEP;
           track[x][y]++; /* Activates point along the track */
         }
       if ( ran > right )
              {
                if ( ran <= down )
                  {
                    x++;     /* Walks down */
                    l_down = YEP;
                    track[x][y]++;
                  } else
                      {
                        y--;   /* Walks left */
                        l_left = YEP;
                        track[x][y]++;
                      }
              }
     } /* END OF PROPABILISTIC DECISION */
   if ( x == 1 )         /*  horizontal upper boundary */
     {
       limit = YES;
       for(i=1;i<=m_mtrx;i++)
         for(j=1;j<=n_mtrx;j++)
           {
             if( track[i][j] == 1 ) {
                val_counter[i][j] += bound[1][y];
                walk_counter[i][j]++;
              }
           }
       l_up = l_right = l_down = l_left = NOP;
     }
   if ( x == m_mtrx )    /* horizontal lower boundary */
     {
       limit = YES;
       for(i=1;i<=m_mtrx;i++)
         for(j=1;j<=n_mtrx;j++)
           {
```

```
                    if( track[i][j] == 1 ) {
                       val_counter[i][j] += bound[m_mtrx][y];
                       walk_counter[i][j]++;
                      }
                  }
                l_up = l_right = l_down = l_left = NOP;
              }
            if ( limit != YES )
              {
                /* if a vertical boundary is reached */
                if ( y == 1 ) { limit = YES;
                for(i=1;i<=m_mtrx;i++)
                  for(j=1;j<=n_mtrx;j++)
                    {
                      if( track[i][j] == 1 ) {
                         val_counter[i][j] += bound[x][1];
                         walk_counter[i][j]++;
                        }
                    }
                  l_up = l_right = l_down = l_left = NOP;
                 }
                if ( y == n_mtrx ) { limit = YES;
                for(i=1;i<=m_mtrx;i++)
                  for(j=1;j<=n_mtrx;j++)
                    {
                      if( track[i][j] == 1 ) {
                         val_counter[i][j] += bound[x][n_mtrx];
                         walk_counter[i][j]++;
                        }
                    }
                  l_up = l_right = l_down = l_left = NOP;
                 }
              }
        } /* end of while */
      } /* END OF Z LOOP (N_WALKS=1,000) */
    } /* END OF l LOOP */
  } /* END OF lz LOOP */

/* RESULT (ARITHMETIC MEAN) OF WALKS IS CALCULATED INCLUDING W-STATS */
max_w = min_w = walk_counter[2][2];
for (lz = 2; lz < m_mtrx; lz++)
  for (l = 2; l < n_mtrx; l++)
    {
    val_counter[lz][l] /= walk_counter[lz][l];
    printf("\nWalks at [%d,%d] : %ld", lz, l, walk_counter[lz][l] );
    if( max_w < walk_counter[lz][l] ) max_w = walk_counter[lz][l];
    if( min_w > walk_counter[lz][l] ) min_w = walk_counter[lz][l];
    }
printf("\n\n");
printf("\nMaximum number of walks performed : %ld", max_w );
printf("\nMinimum number of walks performed : %ld\n\n", min_w );

/* RESULTS ARE PRINTED TO FILE */
/* output in Spyglass ASCII special format Transform manual page 62 */
if ((r_out = fopen( outfile,"wt")) == NULL )  /* ! MODIFY rt  */
  {
    nrerror("Cannot open output file !\n");
    exit( 1 );
```

```
  }
fprintf (r_out,"%d %d\n", m_mtrx, n_mtrx );
fprintf (r_out,"%le %le\n", bound[m_mtrx][1], bound[1][1] );
for (lz = 1; lz <= m_mtrx; lz++)  /* inserting row labels */
  fprintf (r_out, "%le ", z_scale[lz] );
fprintf (r_out, "\n");
for (lz = 1; lz <= n_mtrx; lz++)  /* inserting column labels */
  fprintf (r_out, "%le ", x_scale[lz] );
fprintf (r_out, "\n");

for (lz = 1; lz <= n_mtrx; lz++)  /* appending upper boundary */
  {
    fprintf(r_out, "%le ", bound[1][lz]);
  }
fprintf (r_out, "\n");

for (l = 2; l < m_mtrx; l++) /* writing results */
      {
        fprintf (r_out, "%le ", bound[l][1]);
        for (lz = 2; lz < n_mtrx; lz++)
          {
            fprintf (r_out, "%le ", val_counter[l][lz] );
          }
        fprintf (r_out, "% le", bound[l][n_mtrx]);
        fprintf (r_out,"\n");
      }

for (lz = 1; lz <= n_mtrx; lz++)  /* appending lower boundary */
  {
    fprintf(r_out, "%le ", bound[m_mtrx][lz]);
  }
fprintf (r_out, "\n\n");
fclose( r_out );

/* FREEING UP MEMORY */
free_dmatrix ( val_counter, 1, m_mtrx, 1);
free_ulmatrix ( walk_counter, 1, m_mtrx, 1);
free_imatrix ( track, 1, m_mtrx, 1);

/* END OF FUNCTION */
}
```

```
/*-------------------------------------------------------------------*\
|                                                                     |
| parallel_walk.cs : Random walk routine for the connection machine CM5 |
|                                                                     |
\*-------------------------------------------------------------------*/

#include "mtcarlo.h"
#include <printf.h>

#define NO      0
#define YES     1

#define UP      2   /* markers for previous positions */
#define RIGHT   3
#define DOWN    4
#define LEFT    5

#define DEBUG   0
#define BIT_L 16

static int table_lag = 17, short_lag = 5, width = 32;
static CM_field_id_t *state_field;
static int *residue;

void initialize_random_number_generator (unsigned:current *r)
{  static int seed=45738923; int  i;
   state_field = (CM_field_id_t *) calloc (1, sizeof(CM_field_id_t));
   residue = (int *) calloc (1, sizeof(int));
   CMSSL_init();
   CMSSL_initialize_vp_rng(table_lag, short_lag, width, seed);
   for (i=0;i<10;i++) {
     CMSSL_u_vp_rng_1L (r, 32, 1<<24);
     if (DEBUG) printf ("cmssl_rng_init >> %d %u %u\n",i,  <?=*r, >?=*r);
   }
   *state_field = CMSSL_vp_rng_state_field();
   *residue = CMSSL_vp_rng_residue();
}

void randomize_unsigned_paravar(unsigned:current *r, int bitlength)
{ with (current) {

CMSSL_reinitialize_vp_rng(table_lag,short_lag,width,*state_field,*residue);
    CMSSL_u_vp_rng_1L (r, 32, 1<<bitlength);
    if (DEBUG) printf("randomize_unsigned_paravar :: %u %u\n",<?=*r,>?=*r);
    *state_field = (CM_field_id_t) CMSSL_vp_rng_state_field ();
    *residue = CMSSL_vp_rng_residue();
  }
}

  void
parallel_walk ( unsigned  *inperm,
                float     *inbound,
                char      output_file[MAXSTR],
                int       dK )  /* log permeability differences in input
*/
  {
     /* NORMAL VARIABLES */
```

```
extern unsigned m_mtrx, n_mtrx, u_size;
unsigned long i, j, x, right_ran, max_walks;
unsigned long N_WALKS, min_walks = 0;
FILE *r_out;                        /* output file */

/* PARALLEL VARIABLES AND FUNCTIONS */
float_lut_t boundary_value;
unsigned_lut_t perm;

shape mover_shape;

struct walker {                 /* parallel walker structure */
     unsigned long walks;
     unsigned n_random;
     unsigned up;
     unsigned right;
     unsigned down;
     unsigned left;
     unsigned init_pos;
     short    last_move;
     short    status;           /* flag to distinguish used walkers */
     unsigned curr_pos;
     double   val_counter;
     float    result;
  };

struct walker:mover_shape *mover;

/* FUNCTION PROTOTYPES */
void initialize_random_number_generator (unsigned:current *r);
void randomize_unsigned_paravar(unsigned:current *r, int bitlength);

/* ALLOCATING THE MOVERS */
x = 2;    /* making the axis a power of 2, and at least one bigger */
while ( x < u_size ) x *= 2;
allocate_shape ( &mover_shape, 1, x );
with ( mover_shape )
  {
mover = palloc ( mover_shape, boolsizeof( struct walker:mover_shape ) );

/* SETTING LOOKUP TABLES UP */
perm = make_lookup ( inperm, u_size );
boundary_value = make_lookup ( inbound, u_size );

/* SETTING NUMBER OF WALKS */
N_WALKS = ( 2 * m_mtrx + 2 * n_mtrx - 3 ) * pow( 10, dK );
if ( N_WALKS < 1000 ) N_WALKS = 1000;
if ( N_WALKS > 30000 ) N_WALKS = 5000;

/* SETTING THE STARTING POSITIONS OF THE MOVER'S                    */
/* (the matrix elements at the boundary are excluded by status = NO) */
mover->init_pos = pcoord( 0 );
mover->curr_pos = mover->init_pos;
mover->walks       =   0;
mover->val_counter = 0.0;
mover->status = YES;

where ( mover->init_pos <= n_mtrx ) mover->status = NO;
```

```
      where ( mover->init_pos > (u_size - n_mtrx) ) mover->status = NO;

      j = 1;
      for ( i = (n_mtrx+1); i <= u_size; i++ ) /* x is the length of the mover axis */
        {
              if ( j == n_mtrx )
                {
                  [i]mover->status = NO;
                  j = 1;
                }
              if ( j == 1 )
                {
                  [i]mover->status = NO;
                }
              j++;
        }  /* end init */


      /* PARALLEL OPERATIONS */
      initialize_random_number_generator ( &(mover->n_random) );
      mover->last_move = NO;

      /* setting the minimum number of walks for each starting position */
      while ( min_walks <= 10 ) /* N_WALKS should be here */
        {
          /* restricting movers at the matrix boundary      */
          /* allowing faster walkers to improve precision   */
          /* disallowing a walk to a previous position      */
          where (  mover->status == YES && mover->walks <= ( N_WALKS * 100 ) )
            {
           /* getting move probabilities from all 4 neighbour positions  */
           /* and storing them to make a decision. The probabilities for */
           /* former positions are set to zero                           */
           mover->up    = lookup( perm, (mover->curr_pos-n_mtrx-1) );
           where ( mover->last_move == DOWN ) mover->up = NO;
           mover->right = mover->up + lookup( perm, mover->curr_pos );
           where ( mover->last_move == LEFT ) mover->right = mover->up;
           mover->down  = mover->right + lookup(perm, (mover->curr_pos+n_mtrx-1));
           where ( mover->last_move == UP ) mover->down = mover->right;
           mover->left  = mover->down + lookup( perm, (mover->curr_pos - 2) );
           where ( mover->last_move == RIGHT ) mover->left = mover->down;
           mover->last_move = NO;
           max_walks = >?= mover->walks;
printf("Maximum number of performed walks (single node): %d\n", max_walks );


           /* RANDOM NUMBER GENERATION */
           /* a random number is generated for each mover in the range  */
           /* from 1 to mover->left                                     */
           randomize_unsigned_paravar( &(mover->n_random), BIT_L );
           right_ran = 1;
           while ( right_ran != NO )
             {
                where ( mover->n_random > mover->left )
                  {
                    mover->n_random = mover->n_random / 97.13131313;
                    right_ran = += (int:current) 1;
                  }
```

```
  }

/* DECISION WHERE TO GO */
/* the random number determines where the mover has to go */
where ( mover->n_random <= mover->up )
  {
    mover->curr_pos -= n_mtrx;                /* walks up */
    mover->last_move = UP;
  } else
     {
       where ( mover->n_random <= mover->right )
         {
           mover->curr_pos++;                /* walks right */
           mover->last_move = RIGHT;
         }
       where ( mover->n_random > mover->right )
             {
               where ( mover->n_random <= mover->down )
                 {
                   mover->curr_pos += n_mtrx;/* Walks down */
                   mover->last_move = DOWN;
                 } else
                     {
                       mover->curr_pos--;/* walks left */
                       mover->last_move = LEFT;
                     }
             }
     } /* end of DECISION */

/* STORAGE OF NEW POSITION AND BOUNDARY VALUE */
/* if the mover hits the boundary the walk is over and it has  */
/* to start walking again from the initial position            */

/* mover hits horizontal upper boundary */
where ( mover->curr_pos <= n_mtrx )
  {
     mover->val_counter += lookup(boundary_value, (mover->curr_pos-1));
     mover->walks++;
     /* back to starting position */
     mover->curr_pos = mover->init_pos;
     mover->last_move = NO;
   }
/* mover hits horizontal lower boundary */
where(mover->curr_pos > (u_size-n_mtrx) )
  {
     mover->val_counter +=lookup(boundary_value, (mover->curr_pos-1));
     mover->walks++;
     /* back to starting position */
     mover->curr_pos = mover->init_pos;
     mover->last_move = NO;
   }
/* mover hit lateral boundaries */
x = n_mtrx + 1;
for ( i = 2; i < m_mtrx; i++ )
  {
    where ( mover->curr_pos == x )
      {
         mover->val_counter += lookup( boundary_value,
```

```
                                                (mover->curr_pos-1) );
                    mover->walks++;
                    /* back to starting position */
                    mover->curr_pos = mover->init_pos;
                    mover->last_move = NO;
                  }
                x += ( n_mtrx - 1 );
                where ( x == mover->curr_pos )
                  {
                    mover->val_counter += lookup( boundary_value,
                                                (mover->curr_pos-1) );
                    mover->walks++;
                    /* back to starting position */
                    mover->curr_pos = mover->init_pos;
                    mover->last_move = NO;
                  }
               x++;
              } /* end for */

             /* establishing the local, minimum number of walks to boundary */
             min_walks = <?= mover->walks;

             if ( min_walks > 0 ) printf("min_walks: %d\n", min_walks );

             } /* end where (WALKS * 100) & status restriction  */
             } /* end WHILE min.walks < N_WALKS statement       */

            /* CALCULATING WALK MEANS */
            /* the arithmetic mean of all accumulated boundary values is taken */
            where ( mover->status == YES && mover->val_counter > 0.0 )
               mover->result = mover->val_counter / mover->walks;

     /* FREEING LOOKUP TABLES */
     free_lookup ( perm );
     free_lookup ( boundary_value );


/* ============== END OF MTCARLO SIMULATION ==================== */


   /* RESULTS ARE PRINTED TO FILE */
   /* the results are written in an m x n fashion to an output file */
   /* the boundary values known before are also included therein    */
   if ((r_out = fopen( output_file,"wt")) == NULL )
     {
       printf("\npar_walk: Cannot open output file !\n");
       exit( 1 );
     }
   fprintf (r_out,"CM5-generated potential matrix of size : %d\t %d\n",
            m_mtrx, n_mtrx );
   for (j = 1; j <= n_mtrx; j++)  /* writing upper boundary */
       fprintf(r_out, "%18.3f", inbound[j] );
   fprintf (r_out, "\n");

   /* writing m=2 to m-1 rows of the matrix and boundary values */
   j = 1;
   for (i = (n_mtrx + 1); i <= (u_size - n_mtrx); i++)
         {
```

```
          if ( j == 1 || j == n_mtrx )
            {
              fprintf (r_out, "%18.3f", inbound[i] );
            } else
               {
                 fprintf (r_out, "%18.3f", [i]mover->result );
               }
          if ( j == n_mtrx )
            {
              j = 1;
              fprintf (r_out,"\n");
            }
          j++;
        }

  /* writing lower boundary */
  for (j = (u_size - n_mtrx + 1); j <= u_size; j++)
    {
      fprintf(r_out, "%18.3f", inbound[j]);
    }
  fclose( r_out );

  pfree ( mover );

     }  /* end WITH */


   /* LIBERATING THE PARALLEL MEMORY */
   deallocate_shape ( &mover_shape );

}  /* END function */
```

Appendix E

C-Source Code of the Fluid Tracking Method

The following source code is written in ANSI-C and should compile on any current C-compiler under the UNIX operating system. How the fluid tracking method is applied is explained on the help screen which can be invoked at the beginning of the program. Note that this method performs well only in regions of convergent streamlines. This sufficed the purpose of its design. The method fails to model divergent flow correctly, especially for divergence from node-wide singularities which is explained in Chapter 3, part II.

If this code or modifications thereof are used for numerical modelling the results of which are published, reference should be made in the publication to the originator. This applies, especially to incorporated memory allocation functions which have been published by Press et al., 1988 (Numerical Recipes in C, Cambridge University Press).

The functions `sm_util.c`, `write_double.c` are the same as in the Monte Carlo code. Therefore they are not repeated here.

```
/* ptm.c : FLUID BATCH TRACKING PROGRAM                          */
/*         Sun Version 1.1 for ANSI C-compiler (acc ptm.c -lm)   */
/*         by Stephan K. Matthai 6/7/93                          */

#include "ptm.h"
#include "sm_util.c"      /* memory allocation functions              */
#include "ptm_intr.c"     /* introduces the program                   */
#include "slurp.c"        /* reads rock- types from text-file         */
#include "convert.c"      /* converts 256-color ID's to rocks         */
#include "read_spy.c"     /* reads pressure field and fluid bathces   */
#include "zero.c"         /* zeros' data-array of dimensions i,j,z    */
#include "compare_grad.c" /* compares gradient across adjacent nodes  */
#include "compare_diag.c" /* compares gradient across diagonal nodes  */
#include "stream_walk.c"  /* decision model for streamline migration  */
#include "get_gradient.c" /* x,y- gradient calculation for grid point */
#include "step.c"         /* i,j-steps in specified direction         */
#include "smooth_operator.c" /* smoothes local minima out of solution */
#include "walk.c"         /* migrates along streamline to boundary    */
#include "write_double.c" /* writes smoothed pressure-field, if applic. */

  void
main ()
  {
    extern int m_mtrx, n_mtrx;
    unsigned int i, j, c;
    long secs;
    enum { N, Y } own;
    double ***in_data, fluid;
    char pict_file[MAXSTR], result_file[MAXSTR],
         bound_file[MAXSTR], out_file[MAXSTR],
         patch_file[LARGESTR], dummy[LARGESTR];

    /* EXTERNAL FUNCTION PROTOTYPES */
    void ptm_intro ();
    double ***slurp_picture ( char *infilename );
    void pict_to_rock ( double ***data );
    void read_bound ( char *data_file, double ***data, int data_type );
    void walk ( double ***data, char outfile[MAXSTR], char patch[MAXSTR] );

    /* DATA AQUISITION AND CONVERSION */
    ptm_intro();
    gets( dummy );
    printf("\nMain: Please specify rock-type image [text] file : ");
    gets ( pict_file );
    in_data = slurp_picture ( pict_file );  /* reads rock-types */
    printf("\nMain: Size of image: M = %d; N = %d\n", m_mtrx, n_mtrx );
    pict_to_rock ( in_data );

    /* SOME CONTROL AND FILE KEEPING */
    strncpy ( result_file, pict_file, NAME );
    strcat ( result_file, ".res"); /* extension RES for hydraulic head */
    strncpy ( bound_file, pict_file, NAME );
    strcat ( bound_file, ".flu");  /* extension FLU for fluid amount */
    strncpy ( out_file, pict_file, NAME );
    strcat ( out_file, ".out");    /* extension OUT for results */
    strncpy ( patch_file, pict_file, NAME );
```

```
  strcat ( patch_file, ".patched"); /* ext. PATCHED for smoothed data */

  /* READING OF HYDRAULIC HEAD, AND LOCAL FLUID AMOUNTS */
  read_bound ( result_file, in_data, Z1 );

  /* SOME BOUNDARY CONDITIONS */
  question : { printf ("\nDo you have fluid budget data [Y,N] : ");
               c = getchar();
               if ( c == 89 ) own = Y;
               if ( c == 78 ) own = N;
               if ( c != 89 && c != 78 )
                  {
                    c = getchar();
                    goto question;
                  }
             }
  if ( own )
    {
      read_bound ( bound_file, in_data, Z4 );
    } else {
      printf("\nPlease specify bulk fluid content per rock-node : ");
      scanf("%lf", &fluid );
      for(i=1;i<=m_mtrx;i++)
       for(j=1;j<=n_mtrx;j++) in_data[i][j][Z4] = fluid;
       }

  /* FLUID TRACKING RUN */
  walk ( in_data, out_file, patch_file );

} /* end main */
```

```
/*-------------------------------------------------------------*\
|                                                               |
|   GLOBAL DEFINITIONS for Fluid-Batch Tracking Method          |
|                                                               |
\*-------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <ctype.h>
#include <string.h>

#define F_SOURCES    5          /* Max number of fluids to track      */
#define DIRECTIONS   9          /* Degrees of freedom+1 to walk       */
#define MAXERR     100      /* Max-length error message          */
#define MAXSTR      90          /* max length of filename          */
#define NAME        70          /*  "     "   without extension     */
#define LARGESTR   100          /* max stringlength of screen input   */
#define DI           1.41421356237309505 /* =sqrt2 multipl. factor for  */
                                         /* diagonal elements */

#define ZERO_GRAD    0.00000000000000000 /* no gradient */

#define ZA          10  /* Z-axis of x,y data array */
#define Z1           1  /* MtCarlo solution */
#define Z2           2  /* Permeability */
#define Z3           3  /* Rock Type */
#define Z4           4  /* reservoir capacacity i.e. fluid content */
#define Z5           5  /* fluid source category 1 */
#define Z6           6  /*   "      "       "    2 */
#define Z7           7  /*   "      "       "    3 */
#define Z8           8  /*   "      "       "    4 */
#define Z9           9  /*   "      "       "    5 */
#define Z10         10  /*   "      "       "    6 */

/*
FUNCTION SPECIFICATIONS
=======================
*/

double ***d3matrix(int nrl,int nrh,int ncl,int nch,int vcl,int vch);
void free_d3matrix(double ***m, int nrl,int nrh,int ncl,int nch,int vcl);

int m_mtrx, n_mtrx;          /* globals: size of the picture
matrix         */
```

```
/*-------------------------------------------------------------*\
|                                                               |
| intro.c : introduction to Fluid Batch Tracking Program        |
|                                                               |
\*-------------------------------------------------------------*/

void ptm_intro ()
  {
    int c;
    enum { N, Y } help;

    printf("* ================================================== *\n\n");
    printf("*  *           2d PARTICLE TRACKING METHOD          *  *\n\n");
    printf("*  *      using 256 color-decoding of text files    *  *\n");
    printf("*  *          Vs. 1.1 by S.K. Matthai 9/93          *  *\n\n");
    printf("* ================================================== *\n\n");

    question : { printf ("Type [Y,N] for help (color information etc.): ");
                 c = getchar();
                 if ( c == 89 ) help = Y;
                 if ( c == 78 ) help = N;
                 if ( c != 89 && c != 78 )
                    {
                      c = getchar();
                      goto question;
                    }
               }

    if ( help )
      {
printf("\nSuggested color-to-source rock conversion code (Rainbow
scheme)");
printf("\nwhich is used in the output file :\n\n");
printf("                     silt, slate = yellow -> dataset 1\n");
printf("                         granite = red    -> dataset 2\n");
printf("                      evaporites = white  -> dataset 3\n");
printf("                       limestone = blue   -> dataset 4\n");
printf("              carbonaceous slate = black  -> dataset 5\n");
printf("                porous sandstone = green  -> dataset 6\n\n");
printf("The program will exit if other colors are used!\n\n");

printf("\nThe size of the data-matrices is detected automatically\n");
printf("\nA range of filename extensions for the input as text-files");
printf("\ntells the program which data are available and marks the");
printf("\noutput:\n");
printf("\n[]     = no extension for file which contains the rock type");
printf("\n         matrix. This file is red first.");
printf("\n[.res] = dataset for which the streamfunction is obtained.");
printf("\n[.flu] = dataset with the amount of fluid to be");
printf("\n         dissipated from a specific grid-square");
printf("\n[.out] = output: text-file with 1 dataset for each type of");
printf("\n         fluid which was dissipated.");
printf("\n[.patched] = new streamfunction-dataset which is created when");
printf("\n         local minima or specific noise was encountered in the");
printf("\n         input dataset.\n");

printf("\nThe pressure input (text-file) is expected to have the format of
```

```
a");
printf("\ntext-matrix with the horizontal scale in the first line and");
printf("\nthe value of the vertical scale at the beginning of each line
i.e.");
printf("\nthe Spyglass text-file output format. The scale values are
ignored.");
printf("\nThe rock-type dataset must contain a data matrix [without]");
printf("\nscales. From this input matrix the dimensions of [all] input");
printf("\ndatasets (matrices) are determined.\n");
      }
  } /* end ptm_intro */
```

```
/*-------------------------------------------------------------------------*/
/* slurp.c : picture matrix aquisition routine which checks the size of   */
/*           a matrix presented as [text - integer] input file and reads  */
/*           it in an array. It takes the  input file name and returns a  */
/*           pointer to the array.              */
/*           input file should start with the first data point a[1][1]    */
/*           by Stephan K. Matthai 8/92   */
/*-------------------------------------------------------------------------*/


#include "ptm.h"

/* Vs. 1.0 */

#define IN  1
#define OUT 0

 double
***slurp_picture ( char *infilename )
  {
     int max_m, max_n, i, j, state, m_elements;
     char dummy[MAXSTR];
     double ***data;
     FILE *fp;
     extern int n_mtrx, m_mtrx;
     double ***d3matrix(int nrl,int nrh,int ncl,int nch,int vcl,int vch);
     void nrerror(char error_text[MAXSTR]);
     void zero_data( double ***data, short sets );

      /* CHECKING THE SIZE OF MATRIX */
      if ((fp = fopen ( infilename,"r")) == NULL)
         nrerror("Image file not available !");
      while (!feof(fp))
        {
          state = OUT;
          n_mtrx = m_mtrx = i = j = m_elements = 0;
          while ((i = fgetc( fp )) != EOF )
            {
               if ( isalpha(i) )  /* test file for text */
                 {
                   printf("File contains text !?!\n");
                   printf("Unable to continue reading\n");
                   printf("\nNow exciting...\n");
                   exit ( 1 );
                 }
               if (i == '\n') ++m_mtrx;
               if (i == ' ' || i == '\n' || i == '\t')
                   state = OUT;
               else if (state == OUT)
                 {
                   state = IN;
                   ++m_elements;
                 }
            }
        }
      n_mtrx = m_elements / m_mtrx;
      rewind ( fp );
```

```
    /* READING THE DATA INTO MEMORY */
    data = d3matrix ( 1, m_mtrx, 1, n_mtrx, 1, ZA );
    zero_data( data, ZA );   /* intializing array elemetns to 0.0 */
    for (i = 1; i <= m_mtrx; i++)
      {
        for (j = 1; j <= n_mtrx; j++)
          {
            /* reading the rock-type into storage */
            fscanf( fp,"%lf", &data[i][j][Z3] );
          }
      }
  fclose( fp );
 return ( data );
} /* end slurp.c */
```

```
/*-------------------------------------------------------------------*\
|                                                                     |
| read_spy.c :  reads flagged spyglass text file into data-array      |
|               with scales                                           |
|                                                                     |
\*-------------------------------------------------------------------*/

#include "ptm.h"

  void     /*        filename         storage      entry-position */
read_bound ( char *bound_file, double ***data, int set )
  {
     extern int m_mtrx, n_mtrx;
     int i, j, c;
     double value;
     void nrerror(char error_text[MAXSTR]);
     FILE *b_in;

    if ((b_in = fopen( bound_file,"r")) == NULL )
      nrerror("Data file not found !\n");
    while( (c=fgetc(b_in)) != '\n')  /* scanning row-scale values */
        ;
    for ( i = 1; i <= m_mtrx; i++ )
     {
       fscanf ( b_in, "%lf", &value ); /* scanning column-scale values */
       for ( j = 1; j <= n_mtrx; j++ )
        {
          fscanf ( b_in, "%lf", &value );
          data[i][j][set] = value;
        }
     }
   fclose(b_in);    /* Data are aquired and file is no longer needed */
   printf ("\n\n[%s] has been read successfully\n\n", bound_file );

  }  /* end read_bound */
```

```
/*-------------------------------------------------------------------*\
|                                                                     |
| convert.c   :  Function converts 256 colors (rainbow scheme)       |
|                to rock-type identifiers ranging from 1 to 5         |
|                                                                     |
|                Convention: 1 = evaporite   = yellow                 |
|                            2 = granite     = red                    |
|                            3 = sandstone   = white                  |
|                            4 = limestone   = blue                   |
|                            5 = carb. slate = black                  |
|                                                                     |
|                                                                     |
\*-------------------------------------------------------------------*/

#include "ptm.h"

#define SILTSTONE       1
#define GRANITE         2
#define EVAPORITE       3
#define LIMESTONE       4
#define C_SLATE         5
#define SANDSTONE       6

  void
pict_to_rock ( double ***data )
 {
  int i, j;
  extern int m_mtrx, n_mtrx;

  for (i = 1; i <= m_mtrx; i++)
   {
     for (j = 1; j <= n_mtrx; j++)
      {
        /* COLOR - ROCK-TYPE CONVERSION */
        if ( data[i][j][Z3] == 0.0 ) data[i][j][Z3]        = SILTSTONE;
        else if ( data[i][j][Z3] <= 65.0 ) data[i][j][Z3]  = LIMESTONE;
        else if ( data[i][j][Z3] <= 190.0 ) data[i][j][Z3] = SANDSTONE;
        else if ( data[i][j][Z3] <= 225.0 ) data[i][j][Z3] = EVAPORITE;
        else if ( data[i][j][Z3] <= 254.0 ) data[i][j][Z3] =   GRANITE;
        else if ( data[i][j][Z3] == 255.0 ) data[i][j][Z3] =   C_SLATE;
        else
          {
            printf("\nNot converted: %5.1lf at [%d,%d]",data[i][j][Z3],i,j );
            printf("\nNow exciting...\n\n");
            exit ( -1 );
          }
      }
   }
 } /* end rock-type conversion */
```

```
/*-------------------------------------------------------------*\
|                                                               |
| walk.c   : Fluid dissipation algorithm                        |
|                                                               |
\*-------------------------------------------------------------*/

#include "ptm.h"

 void /* Hydraulic head solution and rock-type, output data filename */
walk ( double ***data, char outfile[MAXSTR], char patched_file[LARGESTR]
)
  {
    /* WALK PARAMETERS */
    int cv, n;                          /* Z-axis definition parameter */
    int x, y;                             /* fast indexes */
    unsigned int i, j, l, lz, z;                        /* move operators */
    unsigned long min = 0;               /* counter for patched minima */
    double ax_x, ax_y;           /* local position in grid square */
    FILE *r_out;

    /* LOGICAL AND RESULT PARAMETERS */
    enum { YES, NO } limit;

    /* EXTERN VARIABLES AND FUNCTIONS */
    extern int m_mtrx, n_mtrx;
    void intro();
    void nrerror(char error_text[MAXERR]);
    void free_d3matrix(double ***m, int nrl,int nrh,int ncl,int nch,int vcl);
    void write_double ( double ***data, char outfile[MAXSTR], int index );

     void
    stream_walk ( short rock,   /* Rock type from which fluid stems */
              double ***data,        /* 3d-data array with solution */
              int idx1,   /* data-element of interest on array axis */
              double *m, double *n,  /* location on square boundary */
              int *i, int *j );            /* coordinates of square */

     int
    smooth_operator( int x, int y,          /* data-array indizes */
                          int idx,/* array axis element of interest */
                    double ***data );       /* 3d-data storage */


  printf("\n*********************************************************\n\n");
  printf("                PARTICLE TRACKING METHOD\n");
  printf("                    (S.K.Matthai 7/93)\n\n");
  printf("*********************************************************\n\n");

    for (lz = 2; lz < m_mtrx; lz++)
      {
        for (l = 2; l < n_mtrx; l++)
          {
           /* SETTING WALK CONDITIONS ON THE BASE OF FLUID SOURCE   */
           cv = (int)data[lz][l][Z3]; /* cv determines which array  */
                                      /* counter is used.           */
           if ( cv < 1 || cv > 6 )    /* Safety */
             {
```

```
            printf("\nWalk: No rock type (%d?) at this point [%d,%d]",
                                cv, lz, l );
            exit ( -1 );
          }

        /* START OF WALKING, N_WALKS DETERMINED BY FLUID CONTENT  */
        if ( data[lz][l][Z4] == ZERO_GRAD )  /* if no fluid there */
          ; else
        for ( z = 1; z <= data[lz][l][Z4]; z++ )
        {
          x = lz;            /* Starting point of fluid migration */
          y = l;
          limit = NO;

          /* SETTING STARTING POSITIONS TO BOTTOM OF GRID SQUARES */
          ax_x = 0.5; ax_y = 0.0;

        /* START INDIVIDUAL WALK */
        while ( limit != YES )
          {
            /* patching potential local minima in solution keeping */
            /* track on how often this is done */
            min += smooth_operator( x, y, Z1, data );

    /* MIGRATING ALONG STREAMLINE AND */
            /* fluid is recorded at starting point in Z5 ff. !     */
            stream_walk ( (cv+4), data, Z1, &ax_x, &ax_y, &x, &y );

            /* CHECKING WHETHER BOUNDARY IS REACHED */
            if ( x == 1 ) limit = YES;
            if ( y == 1 ) limit = YES;
            if ( x == m_mtrx ) limit = YES;
            if ( y == n_mtrx ) limit = YES;
          }  /* end of while = individual walk */
  }  /* FLUID IS EXHAUSTED, z = Z4 */

 if ( data[lz][l][Z4] != ZERO_GRAD )  /* if no fluid there */
   printf("Dissipation complete of fluid %d from [%d,%d]\n",cv,lz,l );

       }      /* END OF INNER FOR LOOP FOR l (columns) */
   }          /* END OUTER lz FOR LOOP (rows) */


 /* RESULTS ARE PRINTED TO FILE */
if ((r_out = fopen( outfile,"wt")) == NULL )
   {
     nrerror("Cannot open output file !\n");
     exit( 1 );
   }
 fprintf (r_out,"Particle tracking results : %s\n", outfile );

 /* THE RESULT ENTRIES ARE WRITTEN IN SEQUENCE */
 for ( n = Z5; n <= ZA; n++ )
   {
     i = n-4;
     fprintf (r_out, "\nFluid type: %d\n", i );
     for (l = 1; l <= m_mtrx; l++)
       {
```

```
          for (lz = 1; lz <= n_mtrx; lz++)
            {
              fprintf (r_out, "%ld ", (unsigned long)data[l][lz][n] );
            }
          fprintf (r_out,"\n");
        }
      fprintf (r_out, "\n");
    } /* END WRITING OF RESULT-SETS */

  fclose( r_out );

  /* if patched more than 10 times solution is printed to file  */
 if ( min > 10 )
   write_double ( data, patched_file, Z1 );

 free_d3matrix ( data, 1, m_mtrx, 1, n_mtrx, 1 );

}    /* END OF FTM */
```

```
/*-------------------------------------------------------------------*/
/* smooth_operator.c : conditional smoothing function which smoothes  */
/*                by averaging a point x,y with its eight neighbours */
/*            only, if this point is a local minimum      */
/*-------------------------------------------------------------------*/

#include "ptm.h"

  int   /* returns int = 1 if hole has been patched          */
smooth_operator( int x, int y,   /* data-array indizes              */
                      int idx,   /* array axis element of interest */
               double ***data )  /* 3d-data storage                */
  {
    register n;
    short max;
    double p[DIRECTIONS], ps, diff;

              /* looking at the solution field */
              /* clockwise, starting at 2. For the diagonal elements */
              /* the gradient is corrected for other distance by DI  */
              ps = data[x][y][idx];
              p[1] = (ps - data[(x-1)][(y+1)][idx]) / DI;
              p[2] = ps - data[x][(y+1)][idx];
              p[3] = (ps - data[(x+1)][(y+1)][idx]) / DI;
              p[4] = ps - data[(x+1)][y][idx];
              p[5] = (ps - data[(x+1)][(y-1)][idx]) / DI;
              p[6] = ps - data[x][(y-1)][idx];
              p[7] = (ps - data[(x-1)][(y-1)][idx]) / DI;
              p[8] = ps - data[(x-1)][y][idx];

             /* comparing neighbourhood of solution points */
             diff = p[1];
             max  = 1;
             for ( n=2; n<DIRECTIONS; n++ ) if ( p[n] > diff ) {
                diff = p[n];
                max = n;
                 }
             /* if local minimum is hit, then this hole */
             /* is mended !, walk steps automatically over shoulder */
             if ( diff <= ZERO_GRAD )
               {
                 printf("\nPatched local minimum at [%d,%d]...\n", x, y );
                 diff = data[x][y][idx];
                 diff += data[(x-1)][(y+1)][idx];
                 diff += data[x][(y+1)][idx];
                 diff += data[(x+1)][(y+1)][idx];
                 diff += data[(x+1)][y][idx];
                 diff += data[(x+1)][(y-1)][idx];
                 diff += data[x][(y-1)][idx];
                 diff += data[(x-1)][(y-1)][idx];
                 diff += data[(x-1)][y][idx];
                 data[x][y][idx]= diff / 9.0;
                 return ( 1 );
               } else return ( 0 );
   } /* end smooth_operator */
```

```
/*-------------------------------------------------------------*/
/* stream_walk.c : maps x  and y (increasing downward) onto  */
/*            new square (Where) to is a short counting       */
/*          clockwise starting at top to define new position.*/
/*-------------------------------------------------------------*/

#include "ptm.h"

#define FROM_ABOVE 1 /* direction from which square is entered */
#define FROM_RIGHT 2
#define FROM_BELOW 3
#define FROM_LEFT 4
#define UP_RI  1
#define LO_RI     2
#define LO_LE  3
#define UP_LE  4
#define RIGHT  1
#define DOWN   2
#define LEFT   3
#define UP     4

/* location on square boundary (m,n) defines from where square   */
/* was entered. New location is set for new square at the end of */
/* the function. ! m,n are returned as zero if walk comes from   */
/* from left or bottom, respectively and vice versa.             */

 void
stream_walk ( short rock,    /* Rock type from which fluid stems */
              double ***data,     /* 3d-data array with solution */
              int idx1,     /* element of interest on array axis */
              double *mp, double *np, /* loc. on square boundary */
              int *ip, int *jp ) /* coordinates of square     */
 {
    double x, y, new; /* x is horizontal increasing to right, y  */
                      /* increasing upward */
    double m, n;
    int i, j, grad, st, status;

      void
    get_gradient( int i, int j, double *x, double *y,
                  int idx, double ***data );

      int
    compare_grad ( int direc, double grad, int i, int j,
                   int idx, double ***data );

      int
    compare_diag(int direc,double x,double y, int i, int j,
                 int idx,double ***data);

      void
    step ( int direc, int *i, int *j );

    /* initial assignments */
    m = *mp; n = *np; i = *ip; j = *jp;
    status = 0;
```

```
  data[i][j][rock] += 1.0;        /* incrementing fluid type-counter */

  /* X,Y VECTOR COMPONENTS ARE OBTAINED */
  get_gradient( i, j, &x, &y, idx1, data );

 /* 1. PATH IF SQUARE IS ENTERED FROM ABOVE */
 if ( n == 1.0 && m != 0.0 && m != 1.0 )
  {
    status++;
    if ( y > 0.0 )          /* if gradient is directed out of square */
      {
        grad = compare_grad ( FROM_ABOVE, y, (i-1), j, idx1, data );
if ( grad == 1 ) {
    if ( x > 0.0 ) { m = 0.0; j++; } else { m = 1.0;j--;}
  } else { n = 0.0; i--; }
      }
    else if ( x == 0.0 )     /* if no horizontal vector component */
      {
        /* sets to from below */
        if ( y > 0.0 ) { i--; n = 0.0; }
        else { i++; n = 1.0; } /* sets to from above */
      }
    else if ( y == 0.0 )        /* if no vertical vector component */
      {
        /* diagonal-up and -right square is new square */
        if ( x > 0.0 ) { m = 0.0; j++; }
        else { m = 1.0; j--; }
      }
    else {                /* if negative vertical vector component and */
        if ( x > 0.0 && y < 0.0 ) /*    positive horizontal component */
          {                /* i.e. move to the right */
            new = n + ((1.0-m) * y/x);
            if ( new > 0.0 ) { n = new; m = 0.0; j++; }
            else if ( new == 0.0 )          /* if migration to corner */
              { m = 0.0; n = 1.0; j++; i++; }
                else {              /* if migration to lower boundary */
                  m += fabs( 1.0/ (y/x) ); n = 1.0; i++; }
          }
        if ( x < 0.0 && y < 0.0 ) /* if negative horizontal component */
          {
            new = n - (m * (y/x));      /* (-) because way to the left */
            if ( new > 0.0 ) { n = new; m = 1.0; j--; }
            else if ( new == 0.0 )/* migration to corner is indicated */
                   { m = 1.0; n = 1.0; j--; i++; }
               else /* migration to lower boundary */
                 { m -= fabs( 1.0/ (y/x) ); n = 1.0; i++; }

          } /* end if negative horizontal component */
      } /* end if negative vertical component */
  } /* end 1. (walk from above) */


  /* 2. PATH IF SQUARE IS ENTERED FROM THE UPPER RIGHT */
  if ( status == 0 && m == 1.0 && n == 1.0 )
    {
      status++;
      if ( y == 0.0 && x < 0.0 ) j--;
      else if ( y == 0.0 && x > 0.0 ) { m = 0.0; j++; }
```

```
            else if ( y == 0.0 && x == 0.0 ) ;
              else if ( y < 0.0 && x == 0.0 ) i++;
              else if ( y > 0.0 && x == 0.0 ) { n = 0.0; i--; }
                else if ( (y/x) == 1.0 ) { i++; j--; }
                   else if ( x < 0.0 && y > 0.0 )
                      {
                      grad = compare_grad ( FROM_ABOVE, y, (i-1),
                                                  j, idx1, data );
  if ( grad == 1 ) { m = 1.0; j--; }
                        else { n = 0.0; i--; }
                       }
                      else if ( x > 0.0 && y < 0.0 )
                         {
                        grad = compare_grad ( FROM_RIGHT, x, i, (j+1),
                                                    idx1, data );
    if ( grad == 1 ) i++;
    else { m = 0.0; j++; }
  }
                       else if ( x > 0.0 && y > 0.0 )
                        {
                          st = compare_diag( UP_RI,x,y,(i-1),(j+1),Z1, data );
if ( st > 0 )
  {
    step( st, &i, &j );
    if ( st == UP ) n = 0.0;
    else if ( st == RIGHT ) m = 0.0;
         } else { m = 0.0; n = 0.0; i--; j++; }
                       }
                       else if ( (y/x) > 1.0 )
                        {
                          m -= 1.0 / (y/x);
                          i++;
                        } else { n -= y/x; j--; }
      } /* END 2. */


   /* 3. PATH IF SQUARE IS ENTERED FROM THE RIGHT */
   if ( status == 0 && m == 1.0 && n != 1.0 && n != 0.0 )
    {
      status++;
      if ( x > 0.0 )  /* if gradient is directed out of square */
        {
          grad = compare_grad ( FROM_RIGHT, x, i, (j+1), idx1, data );
  if ( grad == 1 ) {
      if ( y > 0.0 ) { n = 0.0; i--; } else { n = 1.0; i++; }
    } else { m = 0.0; j++; }
        }
      else if ( y == 0.0 )     /* if no vertical vector component */
        {
          /* in case of horizontal motion */
          if ( x > 0.0 ) { j++; m = 0.0; }
          else { j--; m = 1.0; }        /* sets m to from right */
        }
      else if ( x == 0.0 )   /* if no horizontal vector component */
        {
          /* diagonal-up and -right square is new square */
          if ( y > 0.0 ) { m = 1.0; n = 0.0; i--; }
          else { m = 1.0; n = 1.0; i++; }
```

```
        }
      else {              /* if negative vertical vector component and */
          if ( (y/x) > 0.0 )   /* if positive slope  */
            {
              new = m - (n / (y/x));
              if ( new > 0.0 ) { m = new; n = 1.0; i++; }
              else if ( new == 0.0 )          /* if migration to corner */
                { m = 1.0; n = 1.0; j--; i++; }
                  else {               /* if migration to left boundary */
                    n -= y/x; m = 1.0; j--; }
            }
          if ( (y/x) < 0.0 ) /* if negative slope */
            {
              new = m + (1.0-n) / (y/x); /* (+) because way to the left */
              if ( new > 0.0 ) { m = new; n = 0.0; i--; }
              else if ( new == 0.0 )/* migration to corner is indicated */
                     { m = 1.0; n = 0.0; j--; i--; }
                 else /* migration to left boundary */
                   { n -= y/x; m = 1.0; j--; }

            } /* end else */
         } /* end */
    } /* end 3. (walk from right) */


    /* 4. PATH IF SQUARE IS ENTERED FROM THE LOWER RIGHT */
    if ( status == 0 && m == 1.0 && n == 0.0 )
      {
        status++;
        if ( y == 0.0 && x < 0.0 ) j--;
        else if ( y == 0.0 && x > 0.0 ) { m = 0.0; j++; }
          else if ( y == 0.0 && x == 0.0 ) ;
            else if ( y > 0.0 && x == 0.0 ) i--;
            else if ( y < 0.0 && x == 0.0 ) { n = 1.0; i++; }
              else if ( (y/x) == -1.0 ) { i--; j--; }
                 else if ( x < 0.0 && y < 0.0 )
                     {
                    grad = compare_grad ( FROM_BELOW, y, (i+1), j, idx1, data );
   if ( grad == 1 ) { m = 1.0; j--; }
                       else { n = 1.0; i++; }
                    }
                  else if ( x > 0.0 && y > 0.0 )
                     {
                    grad = compare_grad ( FROM_RIGHT, x, i, (j+1), idx1, data );
    if ( grad == 1 ) i--;
                       else { m = 0.0; j++; }
                    }
                   else if ( x > 0.0 && y < 0.0 )
                    {
                      st = compare_diag( LO_RI,x,y,(i+1),(j+1),Z1, data );
if ( st > 0 )
  {
    step( st, &i, &j );
    if ( st == RIGHT ) m = 0.0;
    else if ( st == DOWN ) n = 1.0;
      } else { m = 0.0; n = 1.0; i++; j++; }
                    }
                   else if ( fabs(y/x) > 1.0 )
```

```
                    {
                      m += 1.0 / (y/x);
                      i--;
                    } else { n += fabs(y/x); j--; }
    } /* END 4. */


   /* 5. PATH IF SQUARE IS ENTERED FROM BELOW */
 if ( status == 0 && n == 0.0 && m != 0.0 && m != 1.0 )
  {
    status++;
    if ( y < 0.0 )  /* if gradient is directed out of square */
      {
        grad = compare_grad ( FROM_BELOW, y, (i+1), j, idx1, data );
if ( grad == 1 ) {
    if ( x > 0.0 ) { m = 0.0; j++; } else { m = 1.0; j--; }
  } else { n = 1.0; i++; }
      }
    else if ( x == 0.0 ) /* if no horizontal vector component */
      {
        /* sets to from above */
        if ( y < 0.0 ) { i++; n = 1.0; }
        else { i--; n = 0.0; } /* sets to from below */
      }
    else if ( y == 0.0 ) /* if no vertical vector component */
      {
        /* right square is new square */
        if ( x > 0.0 ) { m = n = 0.0; j++; }
        else { m = 1.0; n = 0.0; j--; }
      }
    else {               /* if positive vertical vector component and */
        if ( x > 0.0 && y > 0.0 )    /* positive horizontal component */
          {              /* i.e. move to the right */
            new = (1.0-m) * y/x;
            if ( new < 1.0 ) { n = new; m = 0.0; j++; }
            else if ( new == 1.0 )          /* if migration to corner */
              { n = 0.0; m = 0.0; j++; i--; }
                else {              /* if migration to upper boundary */
                  m += 1.0/(y/x); n = 0.0; i--; }
          }
        if ( x < 0.0 && y > 0.0 )  /* if negative slope, left turn */
          {
            new = -(m * (y/x));   /* (-) because way to the left */
            if ( new < 1.0 ) { n = new; m = 1.0; j--; }
            else if ( new == 1.0 )/* migration to corner is indicated */
                   { m = 1.0; n = 0.0; j--; i--; }
               else /* migration to upper boundary */
                 { m -= fabs( 1.0/(y/x) ); n = 0.0; i--; }

          } /* end if negative horizontal component */
       } /* end if negative vertical component */
  } /* end 5. (walk from below) */


  /* 6. PATH IF SQUARE IS ENTERED FROM THE LOWER LEFT */
  if ( status == 0 && m == 0.0 && n == 0.0 )
    {
      status++;
```

```
      if ( y == 0.0 && x > 0.0 ) j++;
      else if ( y == 0.0 && x < 0.0 ) { m = 1.0; j--; }
        else if ( y == 0.0 && x == 0.0 ) ;
          else if ( y > 0.0 && x == 0.0 ) i--;
          else if ( y < 0.0 && x == 0.0 ) { n = 1.0; i++; }
            else if ( (y/x) == 1.0 ) { i--; j++; }
               else if ( x > 0.0 && y < 0.0 )
                  {
               grad = compare_grad ( FROM_BELOW, y, (i+1), j, idx1, data );
if ( grad == 1 ) { m = 0.0; j++; }
                    else { n = 1.0; i++; }
                  }
                 else if ( x < 0.0 && y > 0.0 )
                     {
                 grad = compare_grad ( FROM_LEFT, x, i, (j-1), idx1, data );
   if ( grad == 1 ) i--;
                       else { m = 1.0; j--; }
                     }
                    else if ( x < 0.0 && y < 0.0 )
                      {
                        st = compare_diag( LO_LE,x,y,(i+1),(j-1),Z1, data );
    if ( st > 0 )
  {
    step( st, &i, &j );
    if ( st == DOWN ) n = 1.0;
    else if ( st == LEFT ) m = 1.0;
  } else { m = 1.0; n = 1.0; i++; j--; }
                      }
                    else if ( fabs(y/x) > 1.0 )
                    {
                      m = 1.0 / (y/x);
                      i--;
                    } else { n += y/x; j++; }
    } /* END 6. */
```

```
 /* 7. PATH IF SQUARE IS ENTERED FROM THE LEFT */
 if ( status == 0 && m == 0.0 && n != 1.0 && n != 0.0 )
  {
    status++;
    if ( x < 0.0 )  /* if gradient is directed out of square */
      {
        grad = compare_grad ( FROM_LEFT, x, i, (j-1), idx1, data );
if ( grad == 1 ) {
    if ( y > 0.0 ) { n = 0.0; i--; } else { n = 1.0; i++; }
  } else { m = 1.0; j--; }
      }
    else if ( y == 0.0 ) /* if no vertical vector component */
      {
        /* if drive to the left */
        if ( x < 0.0 ) { j--; m = 1.0; }
        else { j++; m = 0.0; } /* sets m to from left */
      }
    else if ( x == 0.0 ) /* if no horizontal vector component */
      {
        /*  */
        if ( y > 0.0 ) { m = 0.0; n = 0.0; i--; }
        else { m = 0.0; n = 1.0; i++; }
      }
    else {                /* if negative vertical vector component and */
        if ( (y/x) > 0.0 )    /* if positive slope  */
          {                    /* i.e. move to the right */
            new = ((1.0-n) / (y/x));
            if ( new < 1.0 ) { m = new; n = 0.0; i--; }
            else if ( new == 1.0 )          /* if migration to corner */
              { n = 0.0; j++; i--; }
                else {              /* if migration to right boundary */
                  n += y/x; m = 0.0; j++; }
          }
        if ( (y/x) < 0.0 ) /* if negative slope */
          {
            new = fabs ( n / (y/x) );     /* because way to the right */
            if ( new < 1.0 ) { m = new; n = 1.0; i++; }
            else if ( new == 1.0 )/* migration to corner is indicated */
                   { m = 0.0; n = 1.0; j++; i++; }
               else /* migration to right boundary */
                 { n += y/x; m = 0.0; j++; }

          } /* end else */
       } /* end */
  } /* end 7. (walk from right) */


  /* 8. PATH IF SQUARE IS ENTERED FROM THE UPPER LEFT */
  if ( status == 0 && m == 0.0 && n == 1.0 )
    {
      status++;
      if ( y == 0.0 && x > 0.0 ) j++;
      else if ( y == 0.0 && x < 0.0 ) { m = 1.0; j--; }
        else if ( y == 0.0 && x == 0.0 ) ;
          else if ( y < 0.0 && x == 0.0 ) i++;
          else if ( y > 0.0 && x == 0.0 ) { n = 0.0; i--; }
            else if ( (y/x) == -1.0 ) { i++; j++; }
```

```
                else if ( x > 0.0 && y > 0.0 )
                  {
                grad = compare_grad ( FROM_ABOVE, y, (i-1), j, idx1, data );
if ( grad == 1 ) { m = 0.0; j++; }
                    else { n = 0.0; m = 0.0; i--; }
                 }
                else if ( x < 0.0 && y < 0.0 )
                   {
                grad = compare_grad ( FROM_LEFT, x, i, (j-1), idx1, data );
  if ( grad == 1 ) i++;
                     else { m = 1.0; j--; }
                   }
                  else if ( x < 0.0 && y > 0.0 )
                    {
                      st = compare_diag( UP_LE,x,y,(i-1),(j-1),Z1, data );
 if ( st > 0 )
 {
   step( st, &i, &j );
   if ( st == LEFT ) m = 1.0;
   else if ( st == UP ) n = 0.0;
 } else { m = 1.0; n = 0.0; i--; j--; }
                    }
                  else if ( fabs(y/x) > 1.0 )
                   {
                     m = fabs( 1.0 / (y/x) );
                     i++;
                   } else { n += y/x; j++; }
    } /* END 8. */

  /* assignments of return values */
  *mp = m; *np = n; *ip = i; *jp = j;

} /* end stream_walk */
```

```
/*-------------------------------------------------------------*/
/* get_gradient.c : finds meaningful gradient in x,y-direction */
/*                  in matrix-type data set                    */
/*-------------------------------------------------------------*/

#include "ptm.h"

  void
get_gradient( int i, int j, double *x, double *y,
              int idx, double ***data )
 {
    double gx, gy, gx1,gx2, gx3, gy1, gy2, gy3;

    /* initial assignments */
    gx = *x, gy = *y;

    /* weighted gradient is obtained in x-direction */
    gx1 = data[i][(j-1)][idx] - data[i][j][idx];
    gx1 += (data[i][j][idx] - data[i][(j+1)][idx]);
gx1 *= 5.0; /* the central gradient is weighted */
    gx2 = data[(i-1)][(j-1)][idx] - data[(i-1)][j][idx];
    gx2 += (data[(i-1)][j][idx] - data[(i-1)][(j+1)][idx]);
    gx3 = data[(i+1)][(j-1)][idx] - data[(i+1)][j][idx];
    gx3 += (data[(i+1)][j][idx] - data[(i+1)][(j+1)][idx]);
    gx = gx1 + gx2 + gx3;

    /* weighted gradient is obtained in y-direction */
    gy1 = data[(i+1)][j][idx] - data[i][j][idx];
    gy1 += (data[i][j][idx] - data[(i-1)][j][idx]);
gy1 *= 5.0; /* the central gradient is weighted */
    gy2 = data[(i+1)][(j-1)][idx] - data[i][(j-1)][idx];
    gy2 += (data[i][(j-1)][idx] - data[(i-1)][(j-1)][idx]);
    gy3 = data[(i+1)][(j+1)][idx] - data[i][(j+1)][idx];
    gy3 += (data[i][(j+1)][idx] - data[(i-1)][(j+1)][idx]);
    gy = gy1 + gy2 + gy3;

    /* result assignment */
    *x = gx; *y = gy;

 } /* end get_gradient */
```

```
/*-----------------------------------------------------------------*/
/* compare_grad.c : compares gradient at point i,j with gradient   */
/*                  which is input.direc should be 1 if horizontal */
/*                 gradient and 0 if vertical gradient is input    */
/*             It returns 0 for same sign and 1 for opposite  */
/*             sign of gradient.       */
/*-----------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define FROM_ABOVE 1
#define FROM_RIGHT 2
#define FROM_BELOW 3
#define FROM_LEFT 4

  int
compare_grad ( int direc, double grad, int i, int j,
               int idx, double ***data )
{
   extern int m_mtrx, n_mtrx;
   double g;

   if ( grad == 0.0 ) return( 0 );
   else if ( i==1 || j==1 || i==m_mtrx || j==n_mtrx ) return( 0 );
   else if ( direc == FROM_RIGHT )  /* horizontal gradient is tested */
     {
       g = data[i][(j-1)][idx] - data[i][j][idx];
       if ( g <= 0.0 ) return ( 1 );
       else return ( 0 );
     }
   else if ( direc == FROM_LEFT )   /* horizontal gradient is tested */
     {
       g = data[i][j][idx] - data[i][(j+1)][idx];
       if ( g >= 0.0 ) return ( 1 );
       else return ( 0 );
     }
   else if ( direc == FROM_ABOVE )    /* vertical gradient is tested */
     {
       g = data[(i+1)][j][idx] - data[i][j][idx];
       if ( g <= 0.0 ) return ( 1 );
       else return ( 0 );
     }
   else if ( direc == FROM_BELOW )     /* vertical gradient is tested */
     {
       g = data[i][j][idx] - data[(i-1)][j][idx];
       if ( g >= 0.0 ) return ( 1 );
       else return ( 0 );
     }
   else {
       printf("\ncompare_grad: Direction not specified !");
       printf("\n              Now exciting...");
       exit( -1 );
     }
} /* end compare_grad */
```

```
/*-------------------------------------------------------------*/
/* compare_diag.c : compares gradient at point i,j with gradient */
/*                  which is input for diagonal gradients. It    */
/*                  returns 0 for the same sign and 1 for the    */
/*                 opposite.    */
/*-------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define UP_RI 1
#define LO_RI 2
#define LO_LE 3
#define UP_LE 4
#define RIGHT 1
#define DOWN  2
#define LEFT  3
#define UP    4

  int
compare_diag ( int direc, double x, double y,
               int i, int j, int idx, double ***data )
{
   extern int m_mtrx, n_mtrx;
   double gx, gy, diag;

   if ( i==1 || j==1 || i==m_mtrx || j==n_mtrx ) return( 0 );
   else {
   gx = data[i][(j-1)][idx] - data[i][(j+1)][idx];
   gy = data[(i+1)][j][idx] - data[(i-1)][j][idx];

   if ( gx == 0.0 || gy == 0.0 ) return( 0 );
   else
   if ( direc == UP_RI && x > 0.0 && gx < 0.0 && y > 0.0 && gy < 0.0 )
     {
       diag = data[(i+1)][j][idx] - data[i][(j-1)][idx];
       if ( diag >= 0.0 ) return ( UP );
       else return ( RIGHT );
     } /* end if upper right */
   else
   if ( direc == LO_RI && x > 0.0 && gx < 0.0 && y < 0.0 && gy > 0.0 )
     {
       diag = data[i][(j-1)][idx] - data[(i-1)][j][idx];
       if ( diag >= 0.0 ) return ( RIGHT );
       else return ( DOWN );
     } /* end if lower right */
   else
   if ( direc == LO_LE && x < 0.0 && gx > 0.0 && y < 0.0 && gy > 0.0 )
     {
       diag = data[i][(j+1)][idx] - data[(i-1)][j][idx];
       if ( diag >= 0.0 ) return ( LEFT );
       else return ( DOWN );
     } /* end if lower left */
   else
   if ( direc == UP_LE && x < 0.0 && gx > 0.0 && y > 0.0 && gy < 0.0 )
     {
```

```
        diag = data[(i-1)][j][idx] - data[i][(j+1)][idx];
        if ( diag >= 0.0 ) return ( UP );
        else return ( LEFT );
      } /* end if upper left */
    else return ( 0 );

    } /* end security else */
} /* end compare_grad */
```

```
/*-----------------------------------------------------*/
/* step.c : function performs steps on an i,j-lattice */
/*          following directives.                      */
/*-----------------------------------------------------*/

#include <stdio.h>

#define RIGHT 1
#define DOWN  2
#define LEFT  3
#define UP    4

  void
step ( int direc, int *i, int *j )
  {
     int old_i, old_j;

     old_i = *i; old_j = *j;
     if ( direc == RIGHT ) old_j++;
       else if ( direc == DOWN ) old_i++;
         else if ( direc == LEFT ) old_j--;
           else if ( direc == UP ) old_i--;
 else {
                printf("\nstep: Direction not specified !");
                printf("\n      Now exciting...");
                exit( -1 );
              }
     *i = old_i; *j = old_j;

  } /* end step */
```

Appendix F

Implementation of the Electric Analog Model

This appendix contains technical information about the hardware which was designed and built for the data-aquisition from the electric analog board. A second subsection presents the C-source code to drive the data aquisition hardware including a user-interface for the IBM-PC. This code was written by J. Lanc at RSES. The next subsection summarizes the non-standard procedures applied in the preparation of the EA-board. Finally, an example of the scaling procedure for an electric analog model is given.

F.1 Hardware Setup and Technical Specifications

The following summary has been provided by Joseph Lanc who wrote the data aquisition code and built the hardware (Fig. F.1).

The system consists of (Hardware):

- Computer with 80486 processor.
- Data Acquisition Board MIO16H-25
- 3 Multiplexing Boards AMUX-64
- Timing Board TIO-10
- Signal Generator in-house built
- Standard Power Supply +10V/2A
- Matrix Board with 192 test pins
- Resistive Board to be tested.

The hardware is driven by the program `flowsim.exe` written in C using the Borland Turbo C compiler Vs. 3.0 and set of drivers supplied by National Instruments with the MIO16 board. This board has the following specifications:

The system is built around the MIO16 board. This board is configured for 16 single ended inputs, an input range from 0 to 10 Volts, and unipolar input. This gives a resolution of 2.44 mV. Three AMUX-64 boards are daisy-chain connected with the MIO16 via 50 core flat cable. This allocates 12 channels (4 of each AMUX-64) to each channel of the MIO16, giving a total of 192 channels. Each AMUX-64 board is connected via screw terminals to the detector pin array. This board has

Feature	Value	Unit
Number of channels to scan:	192	
Input voltage	10	V
Resolution (12 bits)	2.44	mV
Data acquisition Speed	25	kHz
Time to scan 192 channels	7.68	msec
Data acquisition intervals	0.01–10	sec
Number of samples	$\leq$127	
Output		ASCII or binary

192 test pins each of them spring loaded to ensure perfect contact with the resistive board to be measured. The scanning speed is set to 25 kHz. It can be changed by using an edit option of the Flowsim program (consultation of the manufacturer's manual is necessary). The MIO16 board is configured for external triggering. As soon as a trigger pulse arrives, the board starts taking data out of all 192 channels at a speed of 25 kHz. Then it waits for next pulse to trigger the same procedure. The pulses come from TIO-10 timing board. The pulse frequency (board scanning speed) is set up by programming timers of the TIO-10 board. This is done through the Flowsim software at the prompt "Enter scan interval in msec: ". The board then sends pulses at appropriate intervals to trigger the MIO16 board data acquisition. Data are stored in memory (up to 127 samples) and are then written to disk at the users command for a user-defined file name. This file is in ASCII (default) or in binary format (edit option from the menu).

The board under test is connected with the power supply over the limiter. This is basically a ramp generator, which controls the charging of the board under test. When the breakdown on the board occurs, the circuit senses the increased current and switches down the supply to a desired current until the breakdown terminates. The procedure repeats itself as soon as a high current is detected.

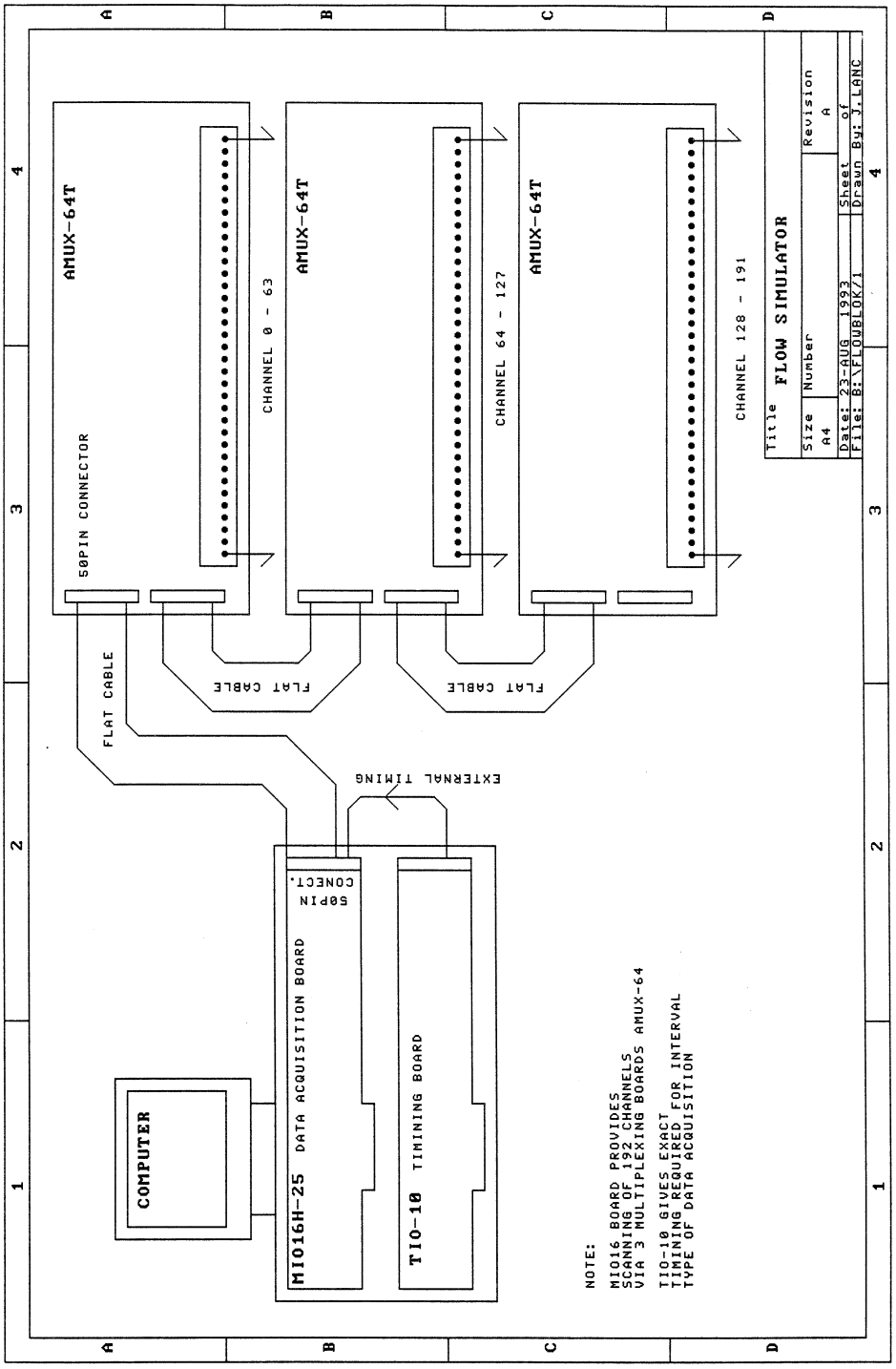


Figure F.1: Data-aquisition hardware set-up of for the electric analog modelling.

F.2 C-Source Code for Data Aquisition

```
/* This program acquires samples from a matrix board of 192 points.
   The board can be sampled up to 127 times in intervals on 10 msec
   to 10 sec and acquisition speed 25kHz.
   It uses National Instruments MIO-16 and TIO-10 boards and three
   AMUX-64 boards.
   Written by J.Lanc (C) 1992 for ANU RSES  */
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <alloc.h>
#include <dos.h>
#include <mem.h>
#include "nidaq.h"          /* function definitions */
#include "nidaqerr.h"       /* error code definitions */

#define TRUE 1;
#define FALSE 0;
#define MIO_16_BRD_CODE 6           /*ID code of MIO 16*/
#define TIO_10_BRD_CODE 14          /*ID code of TIO 10*/
#define MAXCHNS 16

int far *data_buf;      /* pointer to buffer storing acquired data */
int far *ptr;           /* temporary storage */
int temp_buf[193];      /* temporary storage for translation */

struct board { int id, code, baseaddr;     //define board ID
       char name[10];} boards[2];
struct { unsigned int samptime,            //structure of run parameters
      sampint,
      filetype,
      gain;
int           x,y,
      num_chans;} *rp;    //pointer to structure
int maxnum;
char editpar[5][35] = {"Sample time base","Sample interval",
"File type(0=binary)","Number of channels",
"Gain"};
char msg[10][40] = {"None", "MIO_16", "TIO_10"};
char far menu0[10][40] = {"1.Run DAC   ", "2.Edit      ", "3.Save data ",
      "4.View data ", "5.Exit      "};
unsigned int num_pts;    /*actual number of points */
int     numMuxBrds;      /* number of AMUX-64 boards */
int i,               /* loop counter and array index */
brd_code,  /* board code */
err_num,         /* holds NI-DAQ DOS error code */
chans[MAXCHNS],  /* array storing channels */
gains[MAXCHNS],  /* array storing gains */
dum,             /* dummy parameter for the Get_DA_Brds_Info call */
datanotsaved,    /* flag that data not saved */
OK;              /* test true or false */
float freq;              /* scan frequency */
char cr;  /* used to consume carriage return in scanf */
unsigned int num_scan,   /* number of scans during the run*/
temp,retrieved,
```

```
period1, period2;
int timebase,status;
char fname[20];          /* stores the file name */
FILE *data;              /* data file handler */
FILE *file_open(char *p,char c);   /* returns opened file handler */

int menu(char far *c);
void initialize();
int getBoardToUse();               /* procedure for getting the slot/ID */
void printErr(char far *c, int e); /*exit with message in case of error */
void save_data(int far *ip, int n);
void displ_data(int far *pt, int n);
unsigned int reshuffle(int far *source, int n);   /* rearrange data */
void run();
void edit();
void stars();
void welcome();


/*-------------------------------------------------------------------------*/
void main()
{
char ch;
clrscr();
welcome();
initialize();
for (;;) {                      //endless loop
   switch (menu(menu0[0])) {
case '1' :  run(); break;          //run data acq.
case '2' :  edit(); break;
case '3' : save_data(data_buf,num_pts); break;
case '4' : displ_data(data_buf,num_pts); break;
case '5' : ;
case 27  : if (datanotsaved) {
      do {
 gotoxy(24,22);
 printf("Data not saved. Save Y/N ");
 ch = toupper(getch());
 if(ch=='Y' || ch=='N') OK = TRUE;
 } while (!OK);
      if (ch=='Y') save_data(data_buf,num_pts);
      }
   gotoxy(24,24);
   printf("Program terminated");
   exit(1);                        //here exit
}
   }
}
```

```
void welcome()
{
gotoxy(35,6);
printf("W E L C O M E");
gotoxy(30,8);
printf("to Fluid Flow Simulation");
gotoxy(38,10);
printf("Program");
gotoxy(29,16);
printf("Written by J.Lanc (C)1992");
gotoxy(30,22);
printf("Press a key to continue");
_setcursortype(_NOCURSOR);
while (!kbhit()) stars();
_setcursortype(_NORMALCURSOR);
}
```

```
void run()      /* this is the main program to sample the matrix board */
{
unsigned int period;
OK = FALSE;
/* fill the channel and gain arrays */
for (i=0; i<rp->num_chans; i++) {
    chans[i] = i;
    gains[i] = rp->gain;
    }

/* Set the number of the board samples */
while(!OK) {
   printf("\nEnter how many times to sample the board: ");
   scanf("%u",&num_pts);
   if(num_pts>0 && num_pts<128)         //maximum number of scans 127
      OK = TRUE
   else printf("\nValid range 1 to %u\n",(65536/512)-1);
   }
OK = FALSE;
num_pts = num_pts * rp->num_chans;    /* get total number of samples */
if (numMuxBrds)
num_pts = num_pts * numMuxBrds * 4;  //if AMUX board installed
printf("Total number of samples will be %u\n",(num_pts/4)*3);

/*Set the acquisition frequency */
while (!OK) {
   printf("\nEnter scan interval in msec: ");
   scanf("%u",&period);
   if (period>=10 && period<=10000) OK=TRUE
   else printf("\nValid range 10 to 10000\n");
   }
OK = FALSE;
freq = 1.0/period*1000;

/* allocate memory for data */
if((data_buf=farmalloc(num_pts*2))==NULL) {
   printf("\nMemory allocation error.\nAborting..");
   exit(1);
   }
ptr = data_buf;                    //store data buffer pointer
i = 0;
   clrscr();

/* Call the appropriate acquisition procedure */
   err_num = Init_DA_Brds (boards[0].id, &boards[0].code);  //init.board
   if (err_num) printErr("Init_DA_Brds error %d",err_num);
/* configure board to Reference-single-ended, range 0-10V, unipolar */
   err_num = AI_Config(boards[0].id,1,10,1);
   if (err_num) printErr("AI_Config error %d",err_num);
   err_num = AI_Mux_Config(boards[0].id,numMuxBrds);
   if (err_num) printErr("AI_Mux_Config error %d",err_num);
   err_num = DAQ_Config(boards[0].id,1,2);        //external triggering
   if (err_num) printErr("DAQ_Config error %d",err_num);
//   err_num = DAQ_Config(boards[0].id,0,0);      //internal triggering
   if (err_num) printErr("DAQ_Config error %d",err_num);
   err_num = SCAN_Setup(boards[0].id,rp->num_chans,chans,gains);
   if (err_num) printErr("SCAN_Setup error",err_num);
```

```
   CTR_Rate(freq, 0.9, &timebase, &period1, &period2);
   printf("\nReady to begin an acquisition of %u samples ",(num_pts/4)*3);
   gotoxy(20,5);
   printf("Press a key to start.. ");
   getch();
   CTR_Square(boards[1].id,2,timebase,period1, period2);
   err_num = SCAN_Start(boards[0].id,data_buf,num_pts,rp->samptime,
rp->sampint,0,0);
   if (err_num) printErr("SCAN_Start error ",err_num);
   do {
      err_num = DAQ_Check(boards[0].id,&status,&retrieved);
      if (i!=retrieved) {
 gotoxy(30,15);
 printf("Samples: %5u",(retrieved/4)*3);
 i=retrieved;
 }
      if (err_num) printErr("DAQ_Check error ",err_num);
      } while (!status);
CTR_Reset(boards[1].id,2,0);
DAQ_Clear(boards[0].id);          //reset data acquisition board
 //  for (i=0; i<num_pts; i++)        //for testing correct order
 //  *(ptr+i)=i;
printf(".....done.\n");
num_pts = reshuffle(ptr,num_pts);     //rearrange data deleating 4th row
datanotsaved = TRUE;
printf("\n\nPress a key to continue..");
getch();
}
```

```
void printErr (char far *c, int e)
{
gotoxy(10,20);
printf("%s %d",c,e);
exit(1);
}
/*.....................................................................*/
int getBoardToUse ()
/* check that MIO_16 and TIO_10 boards are present in the system */
{
   char cr;
   int dum, err_num, brd, num_found, addr, n;
   num_found = 0;
   for (n=1; n<3; n++) {
      err_num = Get_DA_Brds_Info (n, &brd, &addr, &dum, &dum,
 &dum, &dum, &dum, &dum);
      if (err_num >= 0 && brd==MIO_16_BRD_CODE ) {
   num_found += 1;
   boards[0].id = 1;
   boards[0].code = brd;
   boards[0].baseaddr = addr;
   strcpy(boards[0].name,"MIO-16");
   }
      else if(err_num >=0 && brd==TIO_10_BRD_CODE) {
   num_found += 2;
   boards[1].id = 2;
   boards[1].code = brd;
   boards[1].baseaddr = addr;
   strcpy(boards[1].name,"TIO-10");
   }
      }
   return num_found;
}
```

```
void displf(int e, int t)       /*display one field of edit menu*/
{
if(t)
   textattr(BLACK + (LIGHTGRAY << 4));      //set attribute
else
   textattr(LIGHTGRAY + (BLACK << 4));
gotoxy(rp->x,rp->y+e);
if(e==0) cprintf("%u",rp->samptime);
else if(e==1) cprintf("%u",rp->sampint);
else if(e==2) cprintf("%u",rp->filetype);
else if(e==3) cprintf("%u",rp->num_chans);
else if(e==4) cprintf("%u",rp->gain);
}
/*.....................................................................*/
void edit()                  /*edit run parameters*/
{
unsigned int val;
char c;
clrscr();
OK = FALSE;
printf("\nCheck manual before changing any value. Incorrect values may
cause");
printf("\nyour program crash.   Hit ESC to exit.");
for(i=0; i<maxnum; i++) {         //display all fields
   gotoxy(1,rp->y+i);
   printf("%s",editpar[i]);
   displf(i,0);
   }
i = 0;
displf(i,1);        //display first field
while(!OK) {
   switch (getch()) {
      case 27 : OK = TRUE; break;
      case 0  : switch(getch()) {
  case 72 : displf(i,0); i--; if(i<0) i=maxnum-1; break;
  case 80 : displf(i,0); i++; if(i>maxnum-1) i=0; break;
  }
break;
      case 13 : gotoxy(10,22);
       printf("Enter new value: ");
       scanf("%u",&val);
       if(i==0) rp->samptime = val;
       else if(i==1) rp->sampint = val;
       else if(i==2) rp->filetype = val;
       else if(i==3) rp->num_chans = val;
       else if(i==4) rp->gain = val;
       textattr(LIGHTGRAY + (BLACK << 4));
       gotoxy(10,22);
       clreol();
       break;
     }
      displf(i,1);
      }
textattr(LIGHTGRAY + (BLACK << 4));
}
```

```
unsigned int reshuffle(int far *source, int n)
/* get rid of the garbage in memory */
{
int far *dest, ii;
dest = source;                    //memorize the address
n = n/16;                         //get number of samples per channel
for(i=1; i<n; i++)
   movmem(source+i*16,dest+i*12,24);      //move blocks of data in memory
  //to eliminate garbage from not
  //installed 4th board AMUX-64
for (ii=0; ii<n*12; ii++) {               //eliminate any negative values
   if (*(source+ii)<0) *(source+ii)=0;
   }
return n*12;                              //returns number of samples in mem.
}
/*.....................................................................*/
void displ_data(int far *pt, int n)       /*display data in volts*/
{
int samp,count,ii,far *p;
float volt;
OK = FALSE;
if(data_buf) {
   count = n/192;              //get number of board samples
   samp = 1;
   do {
      while (!OK) {
 printf("\n   %u board samples can be displayed.",count);
 printf("\n   Enter which to view (0=exit): ");
 scanf("%u",&samp);
 if(samp>=0 && samp<=count) OK = TRUE;
 }
 OK = FALSE;
      if(samp) {
 p = pt + 192 * (samp-1);            //calculate offset in buffer
 printf("\nDisplay of board sample %u in volts\n",samp);
 gotoxy(6,wherey());
 for (i=0; i<12; i++) {
    volt = *(p+i)/409.6;
    printf("%1.2f ",volt);
    }
 printf("\n");
 p += 12;
 for (i=0; i<12; i++) {
    for (ii=0; ii<14; ii++) {
       volt = *(p+i*14+ii)/409.6;     //calculate voltage
       printf("%1.2f ",volt);
       }
    printf("\n");
    }
 p += 168;
 gotoxy(6,wherey());
 for (i=0; i<12; i++) {
    volt = *(p+i)/409.6;
    printf("%1.2f ",volt);
    }
 }
      } while(samp);
```

```
   }
else  {
   printf("No data in memory! Press a key.. ");
   getch();
   }
}
```

```
void save_data(int far *ip, int n)      /*save acquired data to disk*/
{
unsigned int num,count,samp;
int delimiter;
float volts;
clrscr();
count = 0;
samp = 1;
delimiter = -1;
if(data_buf) {
   printf("Enter name of data file(no ext.): ");
   scanf("%s",fname);
   if(!rp->filetype) {                //in case binary file required
      strcat(fname,".bin");
      data =file_open(fname,'w');
      i=0;
      for(count=0; count<n; count++) {
 fwrite(ip+count,sizeof(int),1,data);
 i++;
 if(i==192) {
    i=0;
    fwrite(&delimiter,sizeof(int),1,data);
    }
 }
      printf("\n%u samples written to file..press a key ",count);
      }
   else {
      strcat(fname,".asc");
      data = file_open(fname,'t');    //in case ASCII file required
      fprintf(data,"..1..\n");
      for(i=0; i<num_pts; i++) {
 volts = *(ptr+i)/409.6;          //calculate volt values
 fprintf(data,"%2.3f ",volts);
 count++;
 if (count>191) {
    samp++;
    if (i!=num_pts-1) fprintf(data,"\n..%u..\n",samp);
    count = 0;
    }
 }
      printf("\n%u samples written to file..press a key ",i);
      }
   datanotsaved = FALSE;
   close(data);
   }
else printf("\nNo data in memory..  press a key..");
getch();
}
```

```
int menu(char far *c)          /*prints menu and returns picked character*/
{
int n;
clrscr();
for (n=0; n<5; n++) {
   gotoxy(35,8+n);
   printf("%s",c+n*40);
   }
gotoxy(26,16);
printf("Enter number to run: ");
n = getch();
return n;
}
/*.....................................................................*/
FILE *file_open(char *p, char c)
/* opens file for binary reading or writing and returns file pointer */
{
FILE *fp;                                  //open for reading
if (c=='r') {
   if ((fp =fopen(p,"rb"))==NULL) {
      printf("\nCannot open file %s. Press a key ",p);
      getch();
      exit(1);
      }                                   //open for writing
   }
else if (c=='w') {
   if ((fp =fopen(p,"wb"))==NULL) {
      printf("\nCannot open file %s. Press a key ",p);
      getch();
      exit(1);
      }
   }
else if (c=='t') {
   if ((fp =fopen(p,"wt"))==NULL) {
      printf("\nCannot open file %s. Press a key ",p);
      getch();
      exit(1);
      }
   }
return fp;
}
```

```
void stars()
{
int i,k;
textattr(YELLOW + (BLACK << 4));
for (k=1; k<4; k++) {
   i=26;
   do {
      gotoxy(i,4);
      if (k==1) cprintf("*  ");
      else if (k==2) cprintf(" * ");
      else if (k==3) cprintf("  *");
      i += 3;
      } while (i<54);
   i = 4+k;
   do {
      gotoxy(56,i-1);
      printf(" ");
      gotoxy(56,i);
      cprintf("*");
      i += 2;
      } while (i<18+k);
   gotoxy(56,i);
   cprintf(" ");
   i = 52;
   do {
      gotoxy(i,20);
      if (k==1) cprintf("  *");
      else if (k==2) cprintf(" * ");
      else if (k==3) cprintf("*  ");
      i -= 3;
      } while (i>26);
   i = 20 - k;
   do {
      gotoxy(26,i+1);
      cprintf(" ");
      gotoxy(26,i);
      cprintf("*");
      i -= 2;
      } while (i>7-k);
   gotoxy(26,i);
   printf(" ");
   delay(100);
   }
textattr(LIGHTGRAY + (BLACK << 4));
}
```

The following program was used to test potential readings on individual pins of the detector board:

```
/* This program acquires samples from a matrix board of 192 points.
   The board can be sampled up to 127 times in intervals on 10 msec
   to 10 sec and acquisition speed 25kHz.
   It uses National Instruments MIO-16 and TIO-10 boards and three
   AMUX-64 boards.
   The program has been written for ANU RSES by J.Lanc (C) 1992   */
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <dos.h>
#include <mem.h>
#include "nidaq.h"          /* function definitions */
#include "nidaqerr.h"       /* error code definitions */

#define TRUE 1;
#define FALSE 0;
#define MIO_16_BRD_CODE 6            /*ID code */
#define TIO_10_BRD_CODE 14
#define MAXCHNS 16

int data_buf[600];       /* storage for data */

struct board { int id, code, baseaddr;     //define board ID
       char name[10];} boards[2];
struct { unsigned int samprate,            //structure of run parameters
      scanrate,
      gain;
int           x,y,
      num_chans;} *rp;    //pointer to structure
int maxnum;
char editpar[5][35] = {"Sample rate","Sample interval",
"Number of channels","Gain"};
unsigned int num_pts;    /*actual number of points */
int     numMuxBrds;      /* number of AMUX-64 boards */
int i,               /* loop counter and array index */
brd_code,  /* board code */
err_num,         /* holds NI-DAQ DOS error code */
chans[MAXCHNS],  /* array storing channels */
gains[MAXCHNS],  /* array storing gains */
dum,             /* dummy parameter for the Get_DA_Brds_Info call */
OK;              /* test true or false */
float freq;              /* scan frequency */
char cr;  /* used to consume carriage return in scanf */
unsigned int num_scan,   /* number of scans during the run*/
temp,retrieved,
period1, period2;
int timebase,status;
void initialize();
int getBoardToUse();              /* procedure for getting the slot/ID */
void printErr(char far *c, int e); /*exit with message in case of error */
void displ_data(int far *pt);
void displ_all(int far *pt);
void reshuffle(int far *source);   /* rearrange data */
void run();
```

```
void edit();
void welcome();
```

```
void main()
{
char ch;
clrscr();
welcome();
initialize();
while (ch!=27) {
  printf("\nPress key: (V)iew, (R)un, (A)ll-display, (E)dit, ESC=exit: ");
  ch = toupper(getch());
  if(ch=='V')  displ_data(data_buf);
  else if(ch=='R') run();
  else if(ch=='E') edit();
  else if(ch=='A') displ_all(data_buf);
   }
}
/*.....................................................................*/
void welcome()
{
clrscr();
printf("\nThis program will read 3 AMUX-64 board at the rate of 25 kHz
and");
printf("\ndisplay voltage value of a channel that you want to view.");
printf("\nCard MIO-16 must be present in the computer.");
}
/*.....................................................................*/
void initialize()
{
rp->num_chans = 16;
rp->samprate = 10000;
rp->scanrate = 10;
rp->gain = 1;
rp->x = 35;
rp->y = 10;
maxnum = 4;
numMuxBrds = 4;
i = getBoardToUse();   //check what boards installed
if(!i) {
   gotoxy(10,20);
   printf("This program needs MIO_16 board to run");
   gotoxy(10,21);
   printf("No board found. Check whether ATBRDS.CFG file is present");
   gotoxy(10,22);
   printf("in current directory. Exiting..");
   exit(1);
   }
}
```

```
void run()      /* this is the main program to sample the matrix board */
{
OK = FALSE;
/* fill the channel and gain arrays */
for (i=0; i<rp->num_chans; i++) {
    chans[i] = i;
    gains[i] = rp->gain;
    }

/* Set the number of the board samples */
if (numMuxBrds)
num_pts = rp->num_chans * numMuxBrds * 4;  //if AMUX board installed
i = 0;
   clrscr();
/* Call the appropriate acquisition procedure */
   err_num = Init_DA_Brds (boards[0].id, &boards[0].code);  //init.board
   if (err_num) printErr("Init_DA_Brds error %d",err_num);
/* configure board to Reference-single-ended, range 0-10V, unipolar */
   err_num = AI_Config(boards[0].id,1,10,1);
   if (err_num) printErr("AI_Config error %d",err_num);
   err_num = AI_Mux_Config(boards[0].id,numMuxBrds);
   if (err_num) printErr("AI_Mux_Config error %d",err_num);
   err_num = DAQ_Config(boards[0].id,0,0);      //internal triggering
   if (err_num) printErr("DAQ_Config error %d",err_num);
//   err_num = SCAN_Setup(boards[0].id,rp->num_chans,chans,gains);
//   if (err_num) printErr("SCAN_Setup error",err_num);
   printf("\nReady to begin an acquisition ");
   gotoxy(20,5);
   printf("Press a key to start.. ");
   getch();
   err_num = SCAN_Op(boards[0].id,rp->num_chans,chans,gains,data_buf,
     num_pts,rp->samprate,rp->scanrate);
//   err_num = SCAN_Start(boards[0].id,data_buf,num_pts,rp->samptime,
// rp->sampint,0,0);
   if (err_num) printErr("SCAN_Op error ",err_num);
DAQ_Clear(boards[0].id);         //reset data acquisition board
printf(".....done.\n");
//   for (i=0; i<192; i++) *(data_buf+i)=i;
reshuffle(data_buf);      //rearrange data deleating 4th row
}
/*.....................................................................*/
void printErr (char far *c, int e)
{
gotoxy(10,20);
printf("%s %d",c,e);
exit(1);
}
```

```
int getBoardToUse ()
/* check that MIO_16 board is present in the system */
{
   char cr;
   int dum, err_num, brd, num_found, addr, n;
   num_found = 0;
   for (n=0; n<3; n++) {
      err_num = Get_DA_Brds_Info (n, &brd, &addr, &dum, &dum,
 &dum, &dum, &dum, &dum);
      if (err_num >= 0 && brd==MIO_16_BRD_CODE ) {
   num_found = 1;
   boards[0].id = 1;
           boards[0].code = brd;
   boards[0].baseaddr = addr;
   strcpy(boards[0].name,"MIO-16");
   }
      }
return num_found;
}
/*.....................................................................*/
void displf(int e, int t)       /*display one field of edit menu*/
{
if(t)
   textattr(BLACK + (LIGHTGRAY << 4));      //set attribute
else
   textattr(LIGHTGRAY + (BLACK << 4));
gotoxy(rp->x,rp->y+e);
if(e==0) cprintf("%u",rp->samprate);
else if(e==1) cprintf("%u",rp->scanrate);
else if(e==2) cprintf("%u",rp->num_chans);
else if(e==3) cprintf("%u",rp->gain);
}
```

```
void edit()                    /*edit run parameters*/
{
unsigned int val;
char c;
clrscr();
OK = FALSE;
printf("\nCheck manual before changing any value. Incorrect values may
cause");
printf("\nyour program crash.   Hit ESC to exit.");
for(i=0; i<maxnum; i++) {          //display all fields
   gotoxy(1,rp->y+i);
   printf("%s",editpar[i]);
   displf(i,0);
   }
i = 0;
displf(i,1);         //display first field
while(!OK) {
   switch (getch()) {
      case 27 : OK = TRUE; break;
      case 0  : switch(getch()) {
  case 72 : displf(i,0); i--; if(i<0) i=maxnum-1; break;
  case 80 : displf(i,0); i++; if(i>maxnum-1) i=0; break;
  }
break;
      case 13 : gotoxy(10,22);
       printf("Enter new value: ");
       scanf("%u",&val);
       if(i==0) rp->samprate = val;
       else if(i==1) rp->scanrate = val;
       else if(i==2) rp->num_chans = val;
       else if(i==3) rp->gain = val;
       textattr(LIGHTGRAY + (BLACK << 4));
       gotoxy(10,22);
               clreol();
       break;
     }
      displf(i,1);
      }
textattr(LIGHTGRAY + (BLACK << 4));
}
```

```
void reshuffle(int far *source)
{
int far *dest, ii;
dest = source;                  //memorize the address
for (ii=1; ii<16; ii++)         //total size of data is 256
   movmem(source+16*ii,dest+12*ii,24);      //move blocks of data in memory
for (ii=0; ii<192; ii++) {                  //eliminate any negative values
   if (*(source+ii)<0) *(source+ii)=0;
   }
}
/*.....................................................................*/
void displ_data(int far *pt)      /*display data in volts*/
{
int samp,far *p;
OK = FALSE;
p = pt;
clrscr();
if(data_buf) {
   samp = 1;
   do {
      while (!OK) {
 printf("\n       192 channels can be displayed.           ");
 printf("\n       Enter which channel to view (0=exit): ");
 scanf("%u",&samp);
 if(samp>=0 && samp<=192) OK = TRUE;
 }
 OK = FALSE;
      if(samp) {
 gotoxy(wherex(),wherey()-2);
 clreol();
 printf("Channel %u: %1.3f V ",samp,*(p+samp-1)/409.6);
       //  printf("Channel %u: %u ",samp,*(p+samp-1));
 clreol();
 }
      } while(samp);
   }
else  {
   printf("No data in memory! Press a key.. ");
   getch();
   }
clrscr();
}
```

```
void displ_all(int far *pt)
{
int far *p, ii;
OK = FALSE;
p = pt;
clrscr();
if(data_buf) {
   gotoxy(6,wherey());
   for (i=0; i<12; i++) {
      printf("%1.2f ",*(p+i)/409.6);
      }
   printf("\n");
   p += 12;
   for (i=0; i<12; i++) {
      for (ii=0; ii<14; ii++) {
 printf("%1.2f ",*(p+i*14+ii)/409.6);
 }
      printf("\n");
      }
   p += 168;
   gotoxy(6,wherey());
   for (i=0; i<12; i++)
      printf("%1.2f ",*(p+i)/409.6);
   printf("\nPress a key...");
   getch();
   }
else  {
   printf("No data in memory! Press a key.. ");
   getch();
   }
clrscr();
}
```

F.3 Preparation of Electric Analog Models

The electric anolg model is in the process of being patented (Statement 12/11/1993). Permission for the reproduction of electric analog models can be obtained by written request to the author.

The preparation of the two-dimensional electric analog models which were used in the experiments involves three steps:

First, a standard epoxy board is exposed, developed, etched, and drilled so that copper stripes remain at the upper and lower boundary. These serve later as contact bars at which the potential is applied. Circular areas with copper plating should also remain where capacitors, diodes, or cables are to be connected to the board. If these parts are soldered to the copper a much better contact is achieved than with conductive glue. This soldering should occur prior to the next step and the board should be cleaned afterwards with acetone in order to remove any trace of soldering wax. Like in a standard circuit, diodes and capacitors should be attached to the side of the board without the copper plating.

The second step is to apply a conductive coating to the copper-plated side of the board. For this purpose small quantities are mixed of conductor and resistor paste at ratios according to the desired square resistivity. The pastes employed contain graphite dispersed in a thermoplastic rasin. They are supplied by ATCHESON which is an international company based in the UK. The product names are ELECTRODAG 423SS and SS24210 for the conductor and the resistor paste, respectively. The conductive glue which was used is sold as E5620. The company also provides extensive information on mixing ratios, curing and application procedures for the pastes. After the mixing of the pastes the mixture should be diluted by about 50 % to achieve the right viscosity for spray painting. It is important that in contradiction with the manufacturer specifications, BUTYL CELLUSOLVE ACETATE was found to be the only solvent which could be used to dilute the paste mixtures. Other solvents led to paste coagulation. The diluent can be obtained from major chemical suppliers.

It should be assured that the spray paint is stired thoroughly immediately prior to application. Furthermore it has to be applied with a very uniform thickness because the latter has a strong influence on the square resistivity. This is achieved by avoiding drops and spraying multiple thin layers on the board which should be in a horizontal position. A single spray should dry within several hours. When several areas are sprayed with different mixtures, no masking tape should be used, because the adhesive dissolves and mixes with the pastes. This has the effect that carbon particles are moved away from the masked boundary which destroys the uniform conductivity of the model. Instead, a cardboard template can be used which is attached with clams at the sides of the board.

The third preparation step is to cure the thoroughly dried board at 100°C in a well-ventilated oven. This curing which should take at least 10 hours increases the conductivity to a permanent value.

F.4 Example of Scaling of an Electric Analog Model

The model EA1 (Chapter 4, Part II) was designed to represent a geo-hydrologic section with a height of 740 m, a width of 600 m, unit thickness, and a range of storage capacities to be represented by single capacitors. The upper boundary of this model was assigned a depth of 5 km beneath the earth surface. The desired bulk permeability was $10^{-19}\mathrm{m}^2$ i.e. 0.1 μD because the model was aimed at representing a section with a permeability sufficiently low to allow for a fluid pressure buildup to lithostatic pressure at a typical fluid supply rate during contact metamorphism.

After correction for the hydrostatic gradient the driving pressure for flow amounts to

$$\Delta \mathrm{P} = \mathrm{P}_{\mathrm{lith}}(5.7\,\mathrm{km}) - (\mathrm{P}_{\mathrm{hydr}}(5\,\mathrm{km}) + \mathrm{P}_{\mathrm{h}}) \approx 103\mathrm{MPa} \tag{F.1}$$

which gives a vertical pressure gradient ∇P of

$$\nabla \mathrm{P} = \Delta \mathrm{P}/z = 1.39 \times 10^{5}\mathrm{Pa\,m}^{-1} \tag{F.2}$$

For the desired bulk permeability k this gradient is associated with the vertical volume flux q_v

$$q_v = \frac{\mathrm{k}\nabla \mathrm{PA}}{\nu} = 1.39 \times 10^{-11}\mathrm{m}^3\,\mathrm{m}^{-2}\,\mathrm{s}^{-1} \tag{F.3}$$

This calculation is based on a dynamic viscosity of water ν of 0.1 centipoise ($10^{-4}\mathrm{kg\,m}^{-1}\,\mathrm{s}^{-1}$) a value at 400°C, 100 MPa (see discussion Chapter 1, Part II). The flux allows a direct calculation of the desired bulk impedance Z of the EA1 model (cf., model descriptions Chapter 4, Part II)

$$Z = \frac{\Delta V}{q_v wd} = 1.2 \times 10^{9}\Omega \tag{F.4}$$

The bulk capacity is then derived from the storage capacity β_c via the scaling relation between Δh and ΔV and the rock volume represented by the model

$$\kappa = \frac{\Delta h}{\Delta V} = 1.03 \times 10^{7} \tag{F.5}$$

$$C = \frac{\beta_c zwd}{\kappa} \tag{F.6}$$

This capacity needs no further modification for the scaling of the geologic time t with the parameter γ. For instance, to achieve that one model second equals 1 year

$$\gamma = 30,758,400 \tag{F.7}$$

$$Z = Z/\gamma = 39\,\Omega \tag{F.8}$$

For the desired time-scaling and capacity, the desired bulk resistivity of EA 1 was calculated via

$$F_r \simeq 1/Z = \frac{1}{\sqrt{R^2 + \frac{1}{(2\pi/5RC)^2 C^2}}} \tag{F.9}$$

Remember that $1/5RC$ serves as a reasonable approximation of the characteristic frequency f of the circuit. The equation (5.9) was solved for R by Newton Raphson iteration.

If the maximum fluid supply to the model should be 6.52 kg m^{-2} yr^{-1}, and the fluid density is 730 kg m^3 this amounts to a volume flux of 8.93 $\times 10^{-3}$ m^3 and the corresponding current at 1 s (model) = 1 yr (nature) should be

$$I = q_v wd = 0.54A \tag{F.10}$$

which is about twice the current which flows through the board when the fault is inactive. These currents are already quite high. In order to keep the currents and capacities in a range convienient to implement, an additonal downscaling of current and capacity was applied as reflected in the values in the model-description Tables, (Chapter 4, Part II).

The Depositional Environment of the Proterozoic Koolpin Formation, Pine Creek Inlier, Northern Australia: Geochemical Comparison with Modern Shale Sequences

Stephan K. Matthai

Research School of Earth Sciences, Australian National University, G.P.O. Box 4, Canberra, ACT 2601

and

Richard W. Henley

Henley and Etheridge Geoscience Consultants, Deakin House, Suite 4, Geils Court 50, Deakin, ACT 2600

Abstract

The Early to Mid-Proterozoic Koolpin formation forms a lithologically distinct sequence in the sedimentary pile of the Pine Creek Inlier in Northern Australia. This succession hosts several dolerite sills and epigenetic gold deposits like the Cosmo Howley Gold Mine (Matthai et al., 1993). Part of the Koolpin Formation has been referred to as "iron formation" and accorded an "exhalative" deep water origin in much of the recent exploration of the district, with consequent implications for the depositional environment. This paper presents a detailed investigation of the sedimentology, petrography, and geochemistry of the Koolpin Formation at- and in the vicinity of Cosmo Howley. It is shown that the sequence consists mainly of metamorphosed shales with narrow intercalations of formerly carbonate-bearing mudstones in which chert-concretions formed during periods of subaerial weathering. In the hanging wall the sequence grades into highly carbonaceous slates which formerly contained diagenetic dolomite.

Depositional structures indicate that the older part of the Koolpin Formation at Cosmo Howley formed in a low energy, inter- to supra-tidal environment dominated by detrital input of terrestrial providence. The younger carbonaceous slates seem to reflect a transition to shallow marine conditions. The chemical character of the metasediments and their REE-signatures compare well with modern day analogs, if one accounts for potassium enrichment and Ca-, Na-depletion which can be related to fluid flow during mineralization and contact metamorphism. No iron-formations in terms of the definition of James (1954) have been identified at Cosmo Howley, no exhalites occur, and an economic synsedimentary gold-enrichment can be ruled out.

1 Introduction

Compositional variation in the metasedimentary pile has an important impact on metamorphic fluid chemistry, style of deformation, and the location of gold deposits. Motivated by this recognition, the metasedimentary sequence hosting the Cosmo Howley gold deposit has been examined in detail. Results are reported here on the petrography, the sedimentology, and the geochemistry of this succession which is exposed in two open pits. The host rock sequence of the Cosmo Howley Deposit seemed particularly well suited for this type of study, because the gold-hosting part of this stratigraphy has been referred to as "iron formation" and accorded an "exhalative" origin in much of the recent exploration of the district and at Cosmo Howley (Nicholson, 1978; Needham and Roarty, 1980; Goulevitch, 1980; Dunnet and Kavanagh, 1983; Wilkinson, 1982; Nicholson and Eupene, 1984; Alexander et al., 1990), with consequent implications for the depositional environment of the Koolpin Formation. Wilkinson (1982), Dunnet and Kavanagh (1983), and Nicholson and Eupene (1984) have used the localization of the highest gold grades in hornfels next to the hanging wall contact to carbonaceous slates (Fig. 1) to argue for syngenetic i.e. syn-sedimentary gold deposition. Nicholson and Eupene (1984) classified Cosmo Howley as an example of stratiform syngenetic gold deposits which they consider the most important type of gold mineralization in the Pine Creek Inlier.

The objective of is to detail the sedimentology and geochemistry, and thus also the environment of formation of the sedimentary host rock suite at Cosmo Howley. These data provide the background against which the hypothesis of a primary gold enrichment has been tested, and fluid/rock interaction processes associated with contact metamorphism and hydrothermal activity can be understood in relation to controls on gold mineralization. This is especially important in the light of the association between carbonaceous parts of the regional stratigraphy and gold mineralization. This suggests that the chemical heterogeneity and character of the early Proterozoic sedimentary pile was a prime control on the chemistry of fluids migrating through it and/or generated during metamorphic- and magmatic devolatilization of the sediments and granite, respectively. To further isolate the effects of metamorphic devolatilization and alteration, the metasediments are subject of a comparison with modern day sediments. Their formation is well constrained and allows to capture differences that may have been important with respect to gold mineralization.

2 Lithologies

The sedimentary rock sequence exposed in the open pits at Cosmo Howley is about 240 m. thick (Fig. 1). Three dolerite sills divide the sequence into (1) a lowermost carbonaceous metasiltstone, biotite schist and slate interval, (2) a biotitic slate - laminated hornfels interval containing layers of recrystallized chert nodules, and (3) an uppermost carbonaceous slate hornfels interval. The upper interval, within which most of the economic mineralization occurred, contains recrystallized chert and quartz nodule-rich layers in hornfels.

A minimum age constraint for the sedimentary host sequence at Cosmo Howley is provided by the Gerowie Tuff, which disconformably overlies the sequence in the open pit and which has an age of 1885±2 Ma (Needham and Stuart-Smith, 1985).

Walpole et al. (1968), Nicholson and Eupene (1984), and Needham and Stuart-Smith (1985) have correlated the sedimentary host sequence at Cosmo Howley with a thicker succession in the Golden Dyke Dome area (Figs. 1 and 2). This succession forms the Early Proterozoic Koolpin Formation within the South Alligator Group which has been described as a distinctive sequence of iron-rich and tuffaceous sediments (Stuart-Smith et al., 1987). According to Needham and Stuart-Smith(1985), the Koolpin Formation is characterized by ferruginous and carbonaceous phyllite with chert bands, ironstones or banded-iron-formation, and silicified dolomites and marbles. The term "banded-iron-formation" (Needham et al., 1980) was probably applied to the laminated chert-nodular hornfels at Cosmo Howley because of its hematitic character in weathered exposures. Elsewhere, a case was made for an evaporitic origin for some of the Koolpin sediments (Crick and Muir, 1980).

2.1 Petrography and Primary Sedimentary Features

In this study, regional outcrop data and several hundred thin sections, hand specimens from outcrop, costeans and drillcore from the Wildman Siltstone, the Koolpin Formation and the lower part of the overlying Gerowie Tuff were examined from the Australian Geological Service Organization, Canberra regional mapping collection, in addition to sample suites collected from the open pits. These formations are composed of a limited number of alternating lithologies, as described below in order of decreasing abundance. Primary sedimentary features are preserved in many of the lithologies, and these, together with geochemical data, allow comparison to be made with modern sedimentary environments in order to interpret the primary depositional setting and subsequent chemical changes which have been imposed by metamorphism and alteration.

Chloritized biotitic slate (shale ?). — Beds of the chloritized biotitic slate have a massive, finely crystalline appearance and minor compositional banding due to variation in the proportion of biotite and/or of chiastolite porphyroblasts or of quartz. The abundance of partly chloritized biotite varies on a millimeter-scale between 30 and 60 vol.% and is inversely proportional to the abundance of quartz and sericite. The biotite-rich zones may also contain up to 10 vol.% almandine. Bedding planes are perfectly planar and individual beds vary only slightly in thickness along strike for up to 150 m. The slate contains relatively little carbonaceous matter ($\leq$6 vol.%). At the microscope scale, normally-graded intercalations of quartz-silt, less than 1 mm wide, are visible.

In the uppermost 1 m-thick bed of chloritized biotitic slate (sample SM42-3, Fig. 2) over 15 vol.% of the rock consists of spheroidal nodules, 3–25 mm in diameter, with biotite-chloritoid rims. The nodules enclose slate with traces of original bedding but the slate in the nodules is less carbonaceous than the host slate, and up to 80 vol.% of the nodule may be replaced by large andalusite crystals. Quartz-sericite pseudomorphs after large ($\leq$6 mm) rhombohedral crystals of anhydrite or carbonate are visible at the outer margin of the nodules (Fig. 3 A). Both the nodules and the pseudomorphs overgrow traces of the primary texture with a similar layer-spacing as outside of them, so they are both likely to have originated as pore-space-filling-type concretions (cf. von Engelhardt, 1973) largely post-dating sediment compaction. The pseudomorphed crystals contain relict growth zones which indicate that they grew radially outward from the nodule margins (cf., Raiswell, 1971). The iron-rich (molar Fe:Mg = 2:1) biotitic rims of nodules and their alumina-rich, andalusite-bearing, interiors resemble the clay-ironstone concretions (cf., Oertel and Curtis, 1972) that commonly form in shale sequences close to a stationary paleo-surface (Richter, 1983). In shales, abundant concretions with marginal crystals seem to be most common where diagenesis is fast relative to the sediment accumulation rate (Hudson, 1978). Hence, the nodules in the chloritized biotite slate bed may indicate ceasing sedimentation.

Carbonaceous Slate (dolomitic oil shale ?). — The carbonaceous slates bounding the dolerite III sill above and below (Fig. 1) are characterized by structural incompetence, a uniform aphanitic black appearance with rare 1-cm-wide, white sericitic intercalations, and well-defined, planar bedding surfaces at 5–20 cm intervals. In the open pit, bedding-parallel laminae of pyrite are commonly present. This pyrite was introduced during a post-mineralization shear event and is not a primary or syngenetic feature of this lithology.

Three interdigitating sub-types of the carbonaceous slate may be distinguished on the basis of texture. The most common, first type, comprises microlayers of

carbonaceous matter interspaced with lensoidal quartz-sericite-biotite aggregates (Fig. 3 B; samples BMR 79126045, 3SM109-3). The carbonaceous microlayers appear to define truncated asymmetric ripples or scour structures (Fig. 3 D). By visual estimate the total content of carbonaceous matter varies between 15 and 60 vol.%. Chiastolite porphyroblasts (0.3-0.9 mm) are scattered through the slate, and quartz-mica pseudomorphs after dolomite crystals are locally visible (Fig. 3 B). Occasional acicular pseudomorphs up to 2 mm long have haloes enriched in carbonaceous matter and shapes consistent with the habit of anhydrite(sample 3SM108-2, Fig. 3 C). These have been interpreted elsewhere as lutecite (a variety of chalcedony) replacements (Crick and Muir, 1980). The light laminations and macroscopic layers consist of sericite, quartz, biotite and chlorite. Some of the quartz grains preserve detrital features. The second sub-type of the carbonaceous slate contains 40–95 vol.% carbonaceous matter by visual estimate. Laminae of carbonaceous matter, with minor biotite and sericite, in places contain quartz-sericite lenses. Accessory rutile, sphene and tourmaline, and some porphyroblasts of chiastolite and/or garnet are also present.

The third sub-type is a pebble conglomerate in which well-rounded pebbles of carbonaceous slate of the second lithotype are embedded in a matrix of the first. The pebbles are in contact and relatively uniform in size (0.5–3 cm). The finer laminae contain biotite, garnet, and ilmenite. The pebble conglomerate occurs only in the uppermost layer of the carbonaceous slates. The first sub-type has a texture similar to those described from the dolomitic, kerogen-rich "oil shales", typical of evaporite black shale successions in which dolomitization occurred early (Eugster, 1985). In the second sub-type, the carbonaceous microlayers are interpretable as former algal mats from supratidal to intertidal environments. The occasional asymmetric ripples indicate periods of unidirectional water flow.

Biotite-muscovite-schist (argillaceous shale ?). — The biotite-muscovite schist is distinguished from the chloritized biotite slate by the larger average grain size of the preserved detrital quartz-silt fraction (5–20 vol.%), by the abundance of metamorphic biotite and muscovite, and by the presence of albite and abundant large biotite+sericite+quartz pseudomorphs replacing cordierite and K-feldspar ($\leq$7 mm). The schist is poorly cleaved (partly pencil cleavage) but has a centimetre-scale compositional banding and metre-wide layers with well defined boundaries which are locally packed(up to 80 vol.%) with pseudomorphs after cordierite and K-feldspar. The primary grain size varies inversely with the content of mica, which comprises 60–80 vol.% of the rock (muscovite:biotite 2:1 to 1:10). The detrital fraction, where preserved, is poorly sorted, and little carbonaceous matter is present. The compositional bands are truncated in places and thickening and thinning of layers is

common. Minor amounts of recrystallized centimeter-size quartz nodules occur in layers up to 30 cm thick. Because bedding terminates directly against the nodules or else slightly wraps around where it is tangential to the nodules, the nodules are likely to have formed during progressive compaction. The biotite-muscovite schist is likely to have formed from a shale with significant variations in clay content.

Carbonaceous metasiltstone (immature Graywacke ?). — The carbonaceous metasiltstone consists of alternating, mm to cm-thick layers of carbonaceous bands($\geq$40 vol.% carbon), biotite-rich bands ($\leq$50 vol.% biotite) and less carbonaceous ($\leq$15 vol.% carbon) quartz-silt bands with minor muscovite, albite and altered K-feldspar. The detrital fraction is poorly sorted and contains minor rock fragments. Metamorphic chiastolite, rutile and sphene are present mainly in the carbonaceous layers. Stuart-Smith et al. (1980) observed ripple marks and load-casts in the carbonaceous meta-siltstone which they classified as Wildmann formation. Carbonaceous beds of the metasiltstone may also be strongly silicified. The carbonaceous metasiltstone is non-schistose and is further distinguished from the overlying biotite schist by a higher content of carbonaceous matter, a coarser and more abundant silt-sand fraction and more pronounced compositional layering. Depending on the accumulation rate, the mm- to cm-size compositional bands (Fe, Mg, Al-rich vs. SiO_2-rich) may represent a cyclic variation in the maturity of the detrital fraction on a hundred- to thousand-year scale. Typical sedimentation rates are for intracratonic basins, 3-11 mm/1000 years (Schwab, 1976), and for river-sediments $\leq$500 mm/year (Johnson, 1984). Alternatively, these variations could reflect short-term events. e.g., floods or storms. The laminations have been interpreted as varves by Needham et al. (1988).

Laminated quartz-nodular hornfels (silicified lacustrine clay and silt ?). — The laminated quartz-nodular hornfels has a distinctive appearance (Fig. 4) and has been the subject of speculation about primary syn-sedimentary gold enrichment, a banded-iron-formation origin, and a postulated exhalative syn-sedimentary genesis of the gold deposit. The hornfels is composed of chert layers 1–15 cm thick and/or individual nodules of fine-grained recrystallized quartz (Fig. 4) which usually comprise 30-40 vol.% and locally up to 60 vol.% of the rock volume. These are embedded in a matrix of variable composition. Along strike, "quartz blows" occur in which the chert layers or lenses may reach a thickness of 8 m (cf. Nicholson, 1978). The hornfels beds are discontinuous and may pinch out over distances of around 200 m. A typical average thickness for individual beds at Cosmo Howley is 3.5 m. Nodules are concentrated in metre-wide bands in the hornfels beds. As in the biotite-muscovite schist, individual chert nodules are rounded, partly amoeboid in shape and may be indented adjacent to other nodules. The hornfels laminations

both terminate against and wrap slightly around them, indicating that the nodules replaced their country rock during progressive compaction.

The hornfels matrix consists of fine alternations of silt and millimeter-size carbonaceous layers which form asymmetric ripples or fill erosional casts along wiggly erosion surfaces (Fig. 3 D). Polygonal desiccation crack structures, 1 to 3 m across, are present. Randomly oriented small folds and contortions in the chert layers indicate shrinkage of some of the nodular beds after the nodules or their precursors had formed. The hornfels matrix is markedly heterolithic. It is composed of mm- to cm-thick bands of different composition, all of which contain minor quartz-silt intercalations and carbonaceous matter. All compositional bands contain syndepositional listric microfaults (Fig. 3 E) and loop structures, which are common in shales (Potter et al., 1980).

There are three kinds of compositional bands in the hornfels: (1) garnet-rich (almandine-spessartine, $\geq$30 vol.%) with minor green biotite+chlorite ($\leq$20 vol.%), sericite, and quartz; the garnet nucleated in carbonaceous laminae (Fig. 3 F); (2) bands of stellate amphibole partly replacing cordierite (actinolite, cummingtonite, grunerite), and minor chlorite, chloritoid, quartz, and sericite; and (3) bands of predominantly biotite, sericite, and quartz with carbonaceous laminae.

In the garnetiferous bands (1) the carbonaceous matter contains dessication-shrinkage textures similar to those in mudstones where dried mud particles have been incorporated by authigenic carbonate (Fig. 3 B). The amphibole-rich bands (2) are devoid of interpretable primary textures and may represent metamorphosed dolomitic and/or sideritic layers. The biotite-rich bands (3) contain thin, wiggly carbonaceous laminations which are here interpreted as residual structures after original algal mats.

The asymmetric ripple-marks provide evidence for stream or tidal flat deposition. The desiccation textures and the quartz-nodular layers indicate that the sediment dried out periodically, allowing silica concretions to form above the water table. The thickness of individual nodular horizons is consistent with their formation in a weathering profile. The abundant carbonaceous laminae are indicative of a supratidal, intertidal or lacustrine environment. Discontinuous sedimentation and intermittent exposure is represented also by the quartz-nodular hornfels.

Silicified metapelite (Epiclastic volcanic?). — In drillcore, in road cuts along the Stuart Highway, and at the Chinese Howley prospect 3 km northwest of the Cosmo Howley mine, there are silicified, cm-scale laminated and finely crystalline beds with intercalated siltstone lenses. These form part of the Gerowie Tuff, which

disconformably overlies the succession at Cosmo Howley. The silicified layers are frequently contorted or convolutely bedded, as defined by darker laminations. Massive beds, up to 1.5 m thick, are also present. Sixty kilometres northeast of Cosmo Howley, the Gerowie Tuff is an unmetamorphosed, unwelded vitric tuff containing pseudomorphs after K-felspar. Graded bedding and reworking are evident in the laminated beds, indicating aquatic redeposition, which is also evident from thin carbonaceous films between the graded layers. The silt-sand lenses in the tuff are poorly sorted but consist of almost pure quartz. The contorted and convoluted laminations indicate soft-sediment deformation by loading and slumping, potentially indicating an increase in paleo-relief.

Lithological summary. All the lithologies described above have compositions and textures typical of shale (mudstone)-siltstone sequences. The carbonaceous metasiltstone and quartz siltstone intercalations represent the least mature clastic input relative to the biotite schist and the chloritized biotitic slate. The concretionary slate (Fig. 3 A) is the most mature of the sediments. The pronounced cm-scale compositional layers in the carbonaceous metasiltstone, biotite schist and quartz-nodular hornfels may be interpreted as due to seasonal variations in organic activity and clastic input, if high sedimentation rates are assumed, or climatic cycles of decade/century duration.

The lithologies described above can be subdivided on the basis of their carbon content. The first group comprises the biotite schist, the chloritized biotitic slate, quartz siltstone beds, and the silicified tuff. The second encompasses the carbonaceous slates, the carbonaceous metasiltstones and the laminated nodular hornfels. The nodular hornfels and the uppermost layers of the carbonaceous slates contain structures indicative of periodic desiccation and surface exposure during deposition and authigenesis.

3 Stratigraphy

Earlier studies at Cosmo Howley (e.g., Alexander et al., 1990) followed an informal stratigraphy based on interpretation of the original diamond-drilling; however the sequence is now known to have been affected by the Main Pit Shear. The correlation of beds along strike and the assessment of their thickness is complicated by the structural complexity and the strong hydrothermal overprint. The sequence of beds and the thicknesses given here therefore may only be valid in the immediate vicinity of the Cosmo Howley deposit.

Two major discontinuities were found in the sedimentary sequence exposed in

the Cosmo Howley mine (Fig. 1): one is a shear that separates the carbonaceous metasiltstones in the lower part of the sequence from the overlying non-carbonaceous biotite schist; the second is an erosional disconformity which is marked by the pebble conglomerate which divides the uppermost carbonaceous slates from the overlying Gerowie Tuff. Below the tectonic contact, the sediments are characterized by a more abundant coarse-clastic and carbonaceous component than above it. The first laminated quartz-nodular hornfels bed occurs in the uniform chloritized biotitic slate shown in profile 1 (Fig. 2). The quartz-nodular hornfels beds form relatively evenly spaced intercalations in the chloritized biotitic slate. Towards the top of Profile 1 (Fig. 2) the sequence becomes increasingly carbonaceous, and the concretionary slate (SM42-3) forms the top of the chloritized biotitic slate. The uppermost quartz-nodular hornfels overlies thin carbonaceous quartzite beds similar to those in the lower part of the section. In turn it is overlain by a relatively thick uniform package of carbonaceous slate that is intruded by a dolerite sill and in turn overlain by the pebble conglomerate.

A broad comparison with the sequence (Fig. 5) that is exposed in the Golden Dyke Dome area, 20 km southeast of Cosmo Howley, shows that the interval occupied by the quartz-nodular hornfels beds at Golden Dyke is similar in thickness to that at Cosmo Howley, but otherwise no direct stratigraphic correlation can be made. The quartz-nodular hornfels occurs in carbonaceous slate overlain by carbonaceous metasiltstone, i.e., similar to the lower part of the Cosmo Howley stratigraphy. So-called "iron-rich sandstones" are present at the Golden Dyke Dome (Nicholson, 1978), and the chloritized biotitic slate (phyllites in Fig. 5) is much thicker than at Cosmo Howley. The upper Black Shale Member which has been compared to the carbonaceous slates at Cosmo Howley (Walpole et al., 1968; Dunnet and Kavanagh, 1983; Nicholson and Eupene, 1984), is only intermittently carbonaceous at Golden Dyke Dome. The Ashstone member (Fig. 5) is equivalent to the Gerowie Tuff at Cosmo Howley, although dark siliceous beds (hornstone) have not been observed in its vicinity. In contrast to Cosmo Howley, the laminated quartz-nodular hornfels in the Golden Dyke Dome occurs mainly in association with carbonaceous slate. The upper part of the Golden Dyke Dome sequence (Fig. 5) comprises rocks of the overlying "post-orogenic flysch" (Needham et al., 1980) which are not exposed at Cosmo Howley. An equivalent to the lower part of the Cosmo Howley sequence is not exposed at the Golden Dyke Dome. In summary, although the different lithologies present at Cosmo Howley are also found in the Golden Dyke Dome, they have different thicknesses and form different sequences and therefore indicate considerable lateral facies variation.

4 Depositional Environment

The sedimentological data for the Cosmo Howley sequence indicate a transition from an environment dominated by cyclic input of sialic weathering products as relatively coarse clastics, alternating with quiescent intervals of high organic productivity, to one of more uniform shale deposition with decreasing biogenic activity. Subsequently, the depositional environment was repeatedly exposed, allowing silicification above the water table and the formation of the precursor quartz-nodular beds. Organic activity resumed in the wetter periods, and the ripple marks and algal mat textures (i.e., carbonaceous laminations) show that sediments were deposited from water continuously flowing in the same direction. Subsequent transgression is represented by the carbonaceous slate(mudstone) in which dolomite formed during early diagenesis. No indications of evaporite beds or "exhalites" have been found at Cosmo Howley. The carbonaceous slate however is of the oil-shale type which is frequently associated with evaporites (Eugster, 1985). Post-depositional regression and exposure is marked by the upper conglomeratic interval. Finally, pumice ash-flow material was erupted and reworked to form the overlying Gerowie Tuff sequence. This sequence contains sandbars which foreshadowed the onset of flysch deposition and indicate an increase in terrestrial relief and basin subsidence.

The repeated succession chloritized biotitic slate to laminated quartz-nodular hornfels (Fig. 1, 2) may be compared with the present day sediment sequence (Moinik Formation, Fig. 6) at Lake Magadi, Kenya (Hay, 1968). Both sequences formed in shallow water, the latter in an epicontinental lacustrine environment subject to periods of desiccation. Both are dominated by a mudstone-shale detrital fraction, have carbonaceous algal laminations, and are overlain by reworked pyroclastics. At Lake Magadi, the quartz nodules replace precursor layers of magadiite ($NaSi_7O_{13} \times 3H_2O$) and kenyaite ($NaSi_{11}O_{20.5}(OH)_4 \times 3H_2O$) that show the same style of soft-sediment contortion (Eugster, 1969), contraction folds, and cracks due to desiccation as those preserved at Cosmo Howley. At Lake Magadi, desiccation shrinkage and folding happen prior to silicification, during which the material becomes rigid, and this may explain the contortions seen in the chert layers at Cosmo Howley. In this context, low-temperature chert formation at Cosmo Howley is consistent with a heavy oxygen-isotope composition of the nodules and layers (+18 per mil). No gold-enrichment has been observed in the Magadi succession.

5 Lithochemistry

In building an understanding of ore controls and genetic processes at Cosmo Howley, the effect of host-rock composition on mineralization and the relative extent of hydrothermal alteration are important factors to determine. For this purpose several macroscopically least-altered samples (Table 2) were collected and analyzed as representatives of the lithologies present in the pit, for comparison with published shale compositions.

Sample locations are shown in Figures 1 and 2. The samples were slabbed to allow sub-sampling of individual compositional bands prior to major- and trace-element analysis by inductively-coupled plasma (ICP) spectroscopy (Table 2). Quartz nodules were excluded from hornfels samples, so that the analysis represents matrix only. Additional ICP analytical data were also available from earlier unconstrained sampling programmes. A separate suite of samples comprises several thousand ICP analyses obtained by Anaconda Australia Inc. in 1983 from drillcore, most of which penetrated the Main Pit Shear array.

5.1 Geochemical Comparison with Reference Lithologies

The major- and minor-element geochemistry of the different lithologies is described below in order of decreasing age, and comparisons are presented in Figures 7– 9 to illustrate significant distinctions between the Cosmo Howley metasediments and reference samples of unmetamorphosed sediments from similar depositional settings. Minor disparities between the Cosmo Howley carbonaceous slate samples and the reference data may be due partly to high losses on ignition in the former. A very high XRF detection limit restricted the comparison for arsenic (50 ppm).

Chloritized biotitic slate vs. terrigenous shale. — The chloritized biotitic slate (samples SM42-3, 3SM116-1, -2) was compared with an average terrigenous shale (Taylor and McLennan, 1985) which was chosen because of the continental derivation of its clastic fraction. Biotite schist (3SM118-2, SM38-1, 43-1, 45-1) and cordierite- and K-feldspar porphyroblastic schist samples (3SM118-1), which contain more Al_2O_3 and K_2O than the slate, were included in the comparison (Fig. 7 A). The slates and schists are depleted in CaO and Na_2O and enriched in K_2O relative to the reference shale. The chloritized biotitic slate is also enriched in arsenic and boron relative to the reference shale. Carbon contents in the slate are variable but much lower than the other lithologies in the pit. The distinctive low chromium content and the high Al_2O_3/SiO_2 ratio of the concretionary slate (SM42-3, Table 2) distinguish it from the other slates and suggest that it may have originated from an almost pure clay.

Laminated quartz-nodular hornfels vs. Congo/Ganges sediments. — The laminated quartz-nodular hornfels was compared with an average composition for suspended particulate matter in the Congo and the Ganges Rivers, both of which carry mainly terrigenous material (Martin and Meybeck, 1979). The river sediment however does not contain algal mats and has not been subject to authigenic chemical processes or desiccation, whereas the matrix composition of the laminated quartz-nodular hornfels has been modified by silica partitioning to nodules and silica loss to groundwater (see below).

All three compositional types of hornfels (carbonaceous-garnetiferous: samples 3SM-112-2-4; amphibolitic: 3SM113-2-4; carbonaceous-biotitic: 3SM114-1, 115-1) compare poorly with the river sediment for the components shown in Figure 7 B. The river sediments are less carbonaceous than the Cosmo Howley samples. The carbonaceous-biotite hornfels (3SM115-1) has the highest carbon and iron content of all the analyzed samples (Table 2), and all of the hornfels samples have higher carbon contents than the reference. Earlier interpretations of the metasedimentary rocks at Cosmo Howley (Dunnet and Kavanagh, 1983; Needham and Stuart-Smith, 1985; Alexander et al, 1990) have referred to the distinctive hornfels and nodular hornfels units as chemical sediments and "iron formations", implying that they originated from submarine hot springs in deep water environments (Wilkinson, 1982; Nicholson and Eupene, 1984). As shown in Table 2, the total iron content of the matrix, between the chert nodules, is less than 27 wt.% Fe_2O_3, and is more commonly $\leq$11 wt.%. Moreover, as shown in Figure 9, the matrix of nodular- and plain hornfels form two separate populations. The higher iron contents reflect samples from the matrix between diagenetic quartz nodules. These relative iron-enrichment data may be interpreted as due to the following factors:

- Authigenic loss of silica from the matrix during nodule formation (enrichment factor 1.3–1.6).
- Minor iron oxide entrapment during sedimentation. Potentially caused by Fe-oxidation and precipitation due to photosynthetic oxygen from the algal mats (cf. Schidlowsky et al., 1979). This applies only to partially carbonaceous hornfels in which the highest Fe-concentrations were found (Table 2).
- Addition of Fe-carbonate and dolomite consequent on oxidation of organic material to CO_2.
- Concentration increase due to water and CO_2 loss during regional and thermal metamorphism.

SiO_2 compared to the other metasedimentary rocks at Cosmo Howley, the Al_2O_3, TiO_2, and Zr values of the hornfelses are among the lowest (Table 2), but they are still several orders of magnitude higher than those of banded-iron-formations (James, 1954), and therefore indicative of a much higher detrital influx. Iron concentrations in hornfels are the highest observed in the dataset, but some of the iron content may be due to epigenetic sulfides. Recalculation of analyses to account for silica

that was removed to nodules during diagenesis gives an iron content of about 10 wt garnetiferous hornfels. In contrast to the components in Figure 19B, SiO_2 and B contents are comparable with those of the river sediment, but this contrasts with a strong relative depletion of the alkalis in hornfels. The distribution patterns of SiO_2, Fe, the trace elements and Na_2O, K_2O, and boron compare well with the river sediments. In the amphibole hornfels samples (Fig. 7 C), this is however coupled with a strong enrichment of MgO, and CaO, which may be indicative of a former carbonate fraction. The carbonaceous hornfels bands with garnet compare favorably with the terrigeneous shale composite.

There is a pronounced depletion of TiO_2, Cr, V, Na_2O, CaO and an enrichment of K_2O, MgO in the carbonaceous biotite-quartz hornfels bands relative to the river sediments (Fig. 7 B). Correction of the hornfels matrix analysis for quartz nodule formation diminishes the difference in iron content relative to the river sediments but indicates a Na_2O depletion for all three compositional varieties of hornfels.

Carbonaceous slate vs. black shale. — Sedimentologically there is a strong similarity between the modern, shallow-marine black shales (Vine and Tourtelot, 1970) and the carbonaceous slates (metamorphosed dolomitic "oil shales") at Cosmo Howley (3SM108-1, 109-2, 110-2,-3,-5, 111-2). Their compositions are compared in Table 2 and Figure 8. For this comparison, averages of the complete black shale dataset were used because the compositions of these shallow-marine suites do not differ systematically (Vine and Tourtelot, 1970). In spite of the sedimentological analogies, the differences in chemistry are significant. The bulk of the carbonaceous slate is similar only in Fe and MgO content, but it has a strong relative depletion in Na_2O, and a moderate K_2O, As, V and Zr enrichment. In all the samples a minor SiO_2 depletion is coupled with a higher Al_2O_3 content and this may be indicative of either a larger original clay fraction or SiO_2 loss since sedimentation. Apart from the elements shown in Figure 8, a strong relative MnO depletion is present in the carbonaceous slates (Table 2). The same element pattern was found for cm-wide, light-colored, sericitic intercalations in the carbonaceous slate (samples 3SM109-1, 110-4, 111-1; Fig. 8) below the Gerowie Tuff. The trace element abundances of TiO_2 (0.5however, not typical of felsic tuffs.

In spite of the broad similarity of lithologic compositions to those of the reference slate, silica contents are 5-15 wt.% SiO_2 lower and almost all the slate samples have a higher Al_2O_3 relative to the reference shale (Table 2). We interpret this as indicating a consistently higher proportion of clay in the slate precursor sediment, which would be consistent with the proposed delta flood plain or tidal flat depositional environment for the Cosmo Howley area.

Table 1 shows rare-earth-element analyses for the carbonaceous garnetiferous hornfels matrix, sheared pyrite-rich carbonaceous slate and carbonate-altered biotite schist samples. Samples of the garnetiferous hornfels compare relatively well with the epicontinental, carbonaceous Cody shale(Fig. 10) but are relatively depleted in Ce; this indicates the former presence of carbonate (Piper, 1974; Jarvis and Jarvis, 1985). Relative enrichment in the LREE and holmium also occurs. Mobility of the REE during alteration is evident from the occurrence of post-kinematic monazite, xenotime, rutile and apatite in association with fractures and post-gold quartz veins (chapter 3). This is also suggested by one sample of the garnetiferous hornfels (Fig. 10 B) in which a distinctive general enrichment of the REE is present in conjunction with a high gold content (15.5 ppm Au). Similarly, a sample of cordierite-biotite schist which underwent carbonate alteration during late shearing, after gold mineralization (chapter 4), also has a pronounced LREE-enrichment coupled with a negative Ce-anomaly which must reflect the introduction of the calcite into the sample (Fig. 10 C). Its REE total is twice that of the Cody shale and five times that of a pure carbonate (Taylor and McLennan, 1985; Jarvis and Jarvis, 1985). The sheared, pyrite-rich carbonaceous slate shows a general REE enrichment, particularly in the HREE (Fig. 10 D). As the Cody shale contains only a little more than half as much of total REE as the sheared slate with which it otherwise compares well in its sedimentological features, REE introduction during shearing seems a plausible interpretation. REE introduction is also consistent with the greater abundance of apatite, monazite, xenotime, rutile, sphene.

6 Alteration Effects

Figure 11 A shows two populations that differ in their alumina content and their K_2O/Al_2O_3 ratios and reflect chloritized biotitic slate and quartz nodular hornfels samples, respectively. The larger population, A, is characterized by a 97 % correlation of K_2O and Al_2O_3 and strong negative correlations of iron with potassium and alumina. Hornfels samples with $\geq$15 wt.% Fe_2O_3 have less than 15 wt.and 2.5 wt.Boron also occurs in this second population and is indicative of post-gold alteration with tourmaline and apatite.

By contrast, the smaller population (B) is characterized by lower K_2O, and a positive correlation of iron with alumina and potassium and magnesia contents greater than 3.5 wt.%. Samples taken separately from equivalent lithologies within mineralized zones show a greater variance in K_2O/Al_2O_3. Elevated arsenic concentrations, which are representative of syn- and post-gold alteration do not correlate with any

particular potassium or aluminum content.

Analyses of Koolpin samples from elsewhere in the Pine Creek Inlier (Ewers, 1982; Wyborn, 1992, Fig. 23 B) indicate that less altered rocks have a K_2O/Al_2O_3 slope of about 0.33 and intercepts of zero indicating original compositions dominated by potassic clays such as illite. All the K_2O/Al_2O_3 ratios are markedly higher than those of the reference modern shale. By contrast, the populations A and B of the macroscopically least-altered metasedimentary rock suite from Cosmo Howley have positive Al_2O_3 intercepts and higher slopes (0.46). The former are reflected in contact metamorphic andalusite which may replace K-poor clays and the latter mimic the abundance of metamorphic biotite and K-feldspar. We interpret the common K-enrichment relative to the reference samples (Figs. 7– 8) and the high K_2O/Al_2O_3 ratios to indicate that during contact-metamorphism and associated alteration/gold mineralization, potassium was added into biotite and K-feldspar. A similar interpretation may be used to account for the difference between Gerowie Tuff samples from the northeast part of the Pine Creek Inlier (Wyborn, 1992) and those from the aureole of the Burnside Granite (Ewers, 1982). Potassium enrichment, therefore, is an indicator of the large-scale fluid flow processes which occurred during contact-metamorphism. Both biotite and K-feldspar solubility decrease both down temperature and pressure so that the K-enrichment can also be interpreted with respect to the flow direction of the fluids which introduced the potassium. K-introduction is also reflected in K-feldspar vein selvages of gold-bearing QV1 veins in the dolerite at Fleur de Leuse, about 1 km northwest of Cosmo Howley and dolerite at the Bridge Creek Gold Deposit north of Cosmo Howely (Saenger von Oepen et al., 1988).

The hornfels matrix contains the lowest proportion of detrital Fe_2O_3, TiO_2 and Zr, but, as noted above, the highest iron concentrations (Table 6). In the hornfels, TiO_2 values are generally lower than those in the reference rocks, and this may indicate a higher proportion of clay in the sequence because detrital rutile and ilmenite are largely restricted to silt fractions (von Engelhardt, 1940). Throughout the sedimentary sequence titanium correlates negatively with alumina and appears to preserve chemostratigraphic cycles (Fig. 12).

7 Conclusions

The Cosmo Howley sedimentary sequence was deposited in an epicontinental, low-energy, deltaic environment in a semi-arid climate. Through time, in the Cosmo Howley district, the detrital influx became more mature and the depositional environment evolved into a deltaic mudflat with algal mats, parts of which were subject to periodic exposure, during which authigenic processes formed quartz concretions close to the surface. Authigenic reactions coupled with degradation of organic material to carbon dioxide led to the formation of dolomite and anhydrite within the more carbonaceous parts of the stratigraphy. The floodplain was finally transgressed and carbonate-bearing "oil" shales were deposited. A further period of terrestrial weathering led to the formation of a conglomerate which was rapidly covered by reworked felsic epiclastics with intercalated quartz-sand bars. A major basin eventually formed into which turbidites were deposited.

The Cosmo Howley and regional stratigraphy is characterized by large fluctuations in carbon content. The Gerowie Tuff, Burrell Creek, and the Mount Bonnie Formations are largely free of carbonaceous beds, while the Koolpin and the Wildman Formation contain beds which are very carbon-rich. The occurence of diagenetic carbonate was restricted to horizons within the carbonaceous slate and thin intercalations in the quartz nodular hornfels.

The geochemical comparison with modern shales reinforces this interpretation of the sedimentary environment at Cosmo Howley, but also demonstrates that the sequence was subjected to substantial depletion of CaO and Na_2O, and enrichment of K_2O and As and potentially Mn during the large-scale hydrothermal alteration that accompanied contact metamorphism in the aureole of the Cullen Batholith.

Neither deep-marine conditions, nor a "banded-iron-formation" depositional environment, nor syn-sedimentary exhalative activity affected the sedimentary sequence, or played a role in the genesis of the gold deposit. In consequence, the existence of a significant primary gold enrichment seems unlikely. Rather, as is described in the next chapter, the formation of gold-bearing crack-seal quartz veins at Cosmo Howley may be regarded as a consequence of the contact metamorphism and associated dehydration of an aluminous metasedimentary rock sequence.

References

Alexander, K., Kavanagh, M., and Rolfe, G.L. (ed. Hughes, F. (1990). *Geology of the Mineral Deposits of Australia and Papua New Guinea*, volume 1, chapter Cosmo Howley gold deposit, pages 751–753. Australasian Institute of Mining and Metallurgy.

Crick, I. and Muir, M. (1980). Evaporites and uranium mineralization in the Pine Creek Geosyncline. In Ferguson, J. and Goleby, A., editors, *Uranium in the Pine Creek Geosyncline*, pages 531–542, Vienna. IAEA.

Dunnet, D. and Kavanagh, M. (1983). Review of surface and sub–surface geology, I5 stratigraphic unit, Comso Howley anticline, N.T. company report, Anaconda Mining Company Report. unpublished.

Eugster, H. P. (1985). Oil shales, evaporites and ore deposits. *Geochim. Cosmochim. Act.*, 49:619–635.

Ewers, G. (1982). Chemical analyses of Early Proterozoic metasedimentary rocks from the Pine Creek Geosyncline. Record 1882/17, Bureau of Mineral Resources, Canberra , Australia.

Goulevitch, John (eds. Ferguson, J. and Goleby, A. (1980). *Uranium in the Pine Creek Geosyncline. Proceedings of the International Uranium Symposium on the Pine Creek Geosyncline*, chapter Stratigraphy of the Kapalga Formation north of Pine Creek and its relationship to base metal mineralization, pages 307–318. International Atomic Energy Agency, Vienna.

Hudson, J. (1978). Concretions, isotopes, and the diagenetic history of the Oxford clay (Jurassic) of central England. *Sedimentology*, 25:339–370.

James, H. (1954). Sedimentary facies of iron formation. *Econ. Geol.*, 49:235–293.

Jarvis, I. and Jarvis, K. (1985). Rare-earth element geochemistry of standard sediments: a study using inductively coupled plasma spectroscopy. *Chem. Geol.*, 53:335–344.

Martin, J.-M. and Meybeck, M. (1979). Elemental mass-balance of material carried by major world rivers. *Mar. Chem.*, 7:173–206.

Needham, R. and Roarty, M. (1980). *Uranium in the Pine Creek Geosyncline. eds. Ferguson, J. and Goleby, A.B.*, chapter Regional geology of the Pine Creek Geosyncline, pages 1–23. International Atomic Energy Agency, Vienna. Proceedings of the International Uranium Symposium on the Pine Creek Geosyncline.

Needham, R. and Stuart-Smith, P. (1985). Stratigraphy and tectonics of the Early to Middle Proterozoic transition, Katherine – El Sherana area, Northern Territory. *Austral. J. Earth Sci.*, 32:219–230.

Needham, R., Stuart-Smith, P. G., and Page, R. W. (1988). Tectonic evolution of the Pine Creek Inlier, Northern Territory. *Precambr. Res.*, 40/41:543–564.

Nicholson, P. (1978). The geology and economic significance of the Golden Dyke Dome, Northern Territory. Master's thesis, Univ. Queensland.

Nicholson, P. and Eupene, G. (1984). Controls on gold mineralisation in the Pine Creek Geosyncline. In *Proceedings of the Australian Institute of Mining and Metallurgy Conference, Darwin, NT, August 1984*. Australasian Institute of Mining and Metallurgy.

Oertel, G. and Curtis, C. (1972). Clay-ironstone concretions preserving fabrics due to progressive compaction. *Geol. Soc. Amer. Bull.*, 83:2597–2606.

Piper, D. (1974). Rare–earth elements in the sedimentary cycle: A summary. *Geology*, 14:285–304.

Potter, P., Maynard, J., and Pryor, W. (1980). *Sedimentology of shale: Study guide and reference source*. Springer, New York.

Raiswell, R. (1971). The growth of Cambrian and Liassic concretions. *Sedimentology*, 17:147–171.

Richter, D. (1983). Classification of coated grains: discussion. In Peryt, T., editor, *Coated Grains*, pages 7–8. Springer.

Schwab, F. (1976). Modern and ancient sedimentary basins: Comparative accumulation rates. *Geology*, 4:723–727.

von Engelhardt, W. (1940). Die Unterscheidung Wasser- und Wind-sortierter Sande aufgrund der Korngrössenverteilung ihrer leichten und schweren Gemengteile. *Chemie der Erde*, 12:445–465.

von Engelhardt, W. (1973). *Die Bildung von Sedimenten und Sedimentgesteinen*, volume 3. Schweizerbart, Stuttgart.

Walpole, B., Crohn, P., Dunn, P., and Randal, M. (1968). Geology of the Katherine–Darwin region, Northern Territory. *Bur. Mineral. Resour. Australia Bulletin*, 82:304.

A Appendix

A.1 Figures

Figure 1: Summary of the sedimentary rock sequence exposed in the Cosmo Howley open pits.

Figure 2: Detailed section through the part of the Cosmo Howley sequence which hosts most of the gold mineralization. The uppermost hornfels beds has previously been referred to as the I5 stratigraphic unit (Dunnet and Kavanagh, 1983).

Figure 3: Sedimentary structures in the rock sequence exposed at the Cosmo Howley Deposit 1. A - Pseudomorphed anhydrite or carbonate (x) at the margin of a Fe-rich concretion (cm) in the chloritized biotitic slate (sample SM42-3, profile 1 Fig. 2). Note relict growth zoning in the pseudomorph. B - Carbonaceous slate (sample 3SM110-7, Fig. 2). The texture of carbonaceous laminae (cl) interspersed with dolomite nodules (Dol) is typical for a kerogen-rich oil shale. C - Carbonaceous slate (sample 3SM108-2, Fig. 2). Pseudomorph of quartz (possibly recrystallized lutecite, a variety of chalcedony) and sericite replacing anhydrite. The pseudomorph has a halo enriched in carbonaceous matter which could be explained by displacement. D - Asymmetric ripple marks and scour structures in the laminated quartz nodular hornfels (west-Phantom Pit). Note the scour structures (sc) in the straight layer in the lower portion of the photograph. E - Microfault truncating carbonaceous layers in the quartz nodular hornfels. The fault is listric and restricted to individual layers (white mineral is contact metamorphic almandine-garnet). F - Garnetiferous band in the carbonaceous hornfels (sample BMR 79126045). Texture is analogous to dried out particulate mud with carbonate infills in the dessication fractures. The carbonate is now replaced by quartz, serizite, and chlorite.

Figure 4: Laminated quartz nodular hornfels from 3 km south of Cosmo Howley. The volumetrically large proportion of quartz nodules and bands is intensly folded and the fold axes have random orientations. Some layers have been juxtaposed on each other due to dessication shrinkage. Note also the discontinuous nature of the chert bands and their thickness variations.

Figure 5: The Golden Dyke stratigraphic sequence, 20 km south-east of Cosmo Howley (reconstructed from Nicholson, 1978). The sequence was not recorded in as much detail as that at Cosmo Howley, but a broad comparison can still be made, although contact metamorphism in the Golden Dyke dome area is more pronounced. A bar(hatchured) marks where a correlation to Cosmo Howley was previously proposed by Walpole et al. (1968) and Nicholson (1978).

Figure 6: Lithostratigraphic section of the Moinik formation at Lake Magadi, Kenia (after Hay, 1968).

Figure 7: Geochemical comparison of Cosmo Howley metasedimentary rocks and modern sediments from the Congo and Ganges Rivers; A - Major and trace elements in slate and schist vs. average terrigenous shale of Taylor and McLennan, 1985 ('T & M'), B - Major and trace elements in carbonaceous garnetiferous hornfels excluding quartz nodules vs. Congo and Ganges river sediments of Martin and Meybeck, 1979 ('M & M'), C - Major and trace elements in amphibolitic hornfels excluding quartz nodules vs. average Congo and Ganges river sediments. D - Major and trace elements in carbonaceous biotitic hornfels excluding quartz nodules vs. Congo and Ganges river sediments of Martin and Meybeck.

Figure 8: Geochemical comparison of Cosmo Howley metasedimentary rocks and well characterized black shales; A - Major and trace elements in carbonaceous slate vs. black shale, B - Major and trace elements in light sericitic intercalations of the carbonaceous slate vs. average black shales of Vine and Tourtelot, 1970 ('V & T').

Figure 9: Schematic representation of relative iron-enrichment of nodular hornfels matrix due to authigenesis and diagenesis. M signifies the compositional range of typical mudstones ('M & M') in modern analogues of the Cosmo Howley sedimentary environment. Small increases in iron-content due to authigenic carbonate formation are amplified by silica nodule formation and later loss of volatiles during metamorphism.

Figure 10: Geochemical comparison of rare earth-element patterns of Cosmo Howley metasedimentary rocks and a well characterized reference shale. The Appalachian Cody Shale SCo-1 was chosen as reference shale, following the suggestion of Jarvis and Jarvis (1985). The NASC standard was rejected because it comprises non-shale samples and others whose derivation is unclear (Gromet et al., 1984). Because Jarvis and Jarvis (1985) did not measure Tb, a value was substituted from the terrigenous shale of Taylor and McLennan (1985) . A - Matrix of macroscopically unaltered laminated quartz-nodular garnet hornfels. B - Mineralized matrix of altered quartz-nodular garnet hornfels. C - Biotite schist with hydrothermal calcite. D - Sheared carbonaceous slate with abundant epigenetic pyrite.

Figure 11: A - $Al_2O_3 - K_2O$ relationship for metasedimentary rocks at Cosmo Howley, m = atomic K/Al-ratio; B - comparison of $Al_2O_3 - K_2O$ relationships of metasedimentary rocks at Cosmo Howley with modern shales and mudstones and equivalent lithologies in the Pine Creek Inlier (data from Ewers, 1982).

Figure 12: Moving averages at 2.5 m. intervals for calcium, magnesium, iron and titanium from Cosmo Howley drillcore, showing the preservation of chemostratigraphic cycles. Note sympathetic calcium and iron, titanium; antipathetic to iron and mixed correlation of magnesium. The trace element data were obtained from 0.5 m. sample intervals using ICP (Anaconda, Australia) laboratory, 1983). The missing intervals correspond to dolerite.

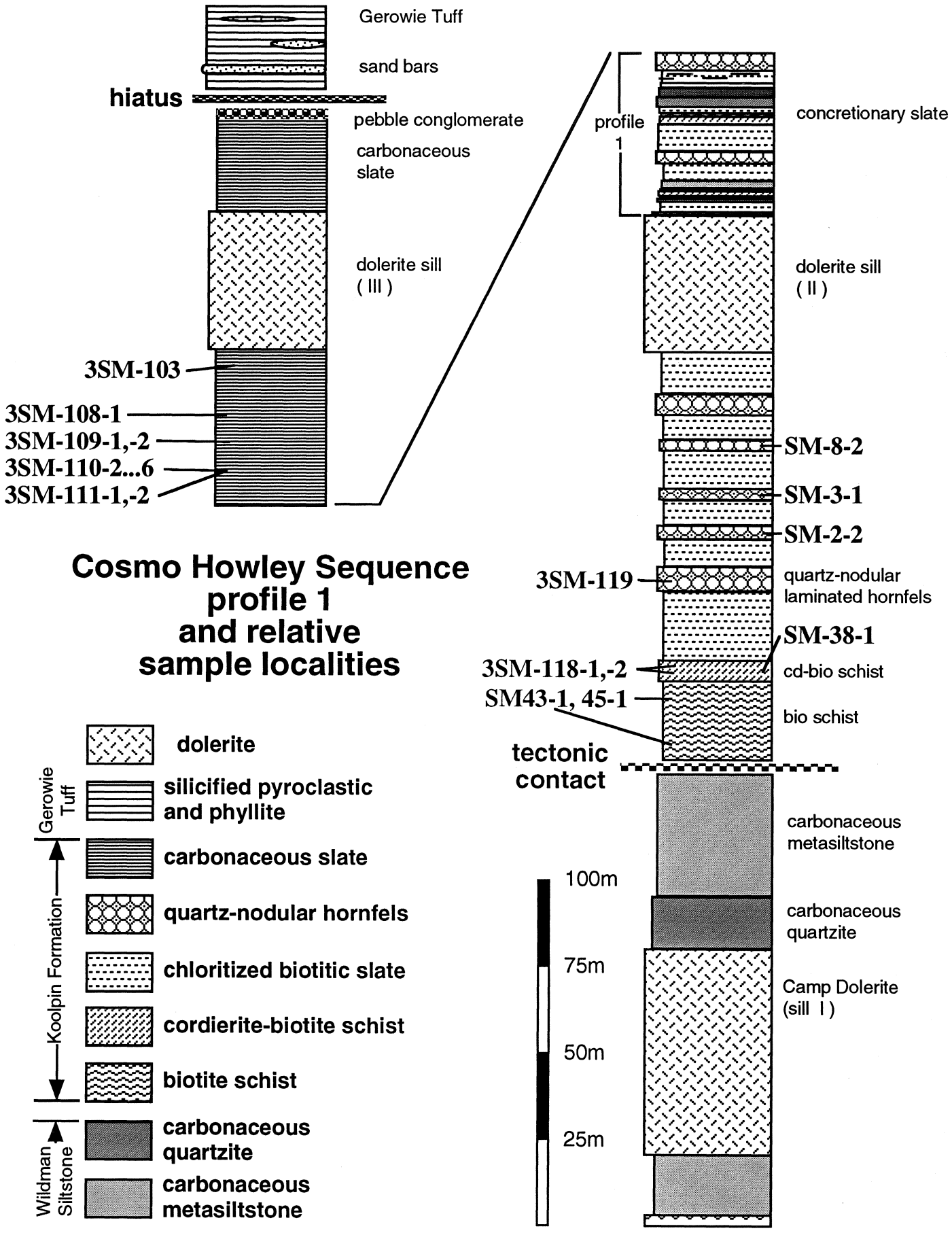

Gerowie Tuff
sand bars
hiatus
pebble conglomerate
carbonaceous slate
dolerite sill (III)
3SM-103
3SM-108-1
3SM-109-1,-2
3SM-110-2...6
3SM-111-1,-2
profile 1
concretionary slate
dolerite sill (II)
SM-8-2
SM-3-1
SM-2-2
3SM-119
quartz-nodular laminated hornfels
SM-38-1
3SM-118-1,-2
cd-bio schist
SM43-1, 45-1
bio schist
tectonic contact
carbonaceous metasiltstone
carbonaceous quartzite
Camp Dolerite (sill I)
100m
75m
50m
25m
Cosmo Howley Sequence profile 1 and relative sample localities
dolerite
silicified pyroclastic and phyllite
carbonaceous slate
quartz-nodular hornfels
chloritized biotitic slate
cordierite-biotite schist
biotite schist
carbonaceous quartzite
carbonaceous metasiltstone
Gerowie Tuff
Koolpin Formation
Wildman Siltstone

Figure 1

Cosmo Howley sequence profile 1

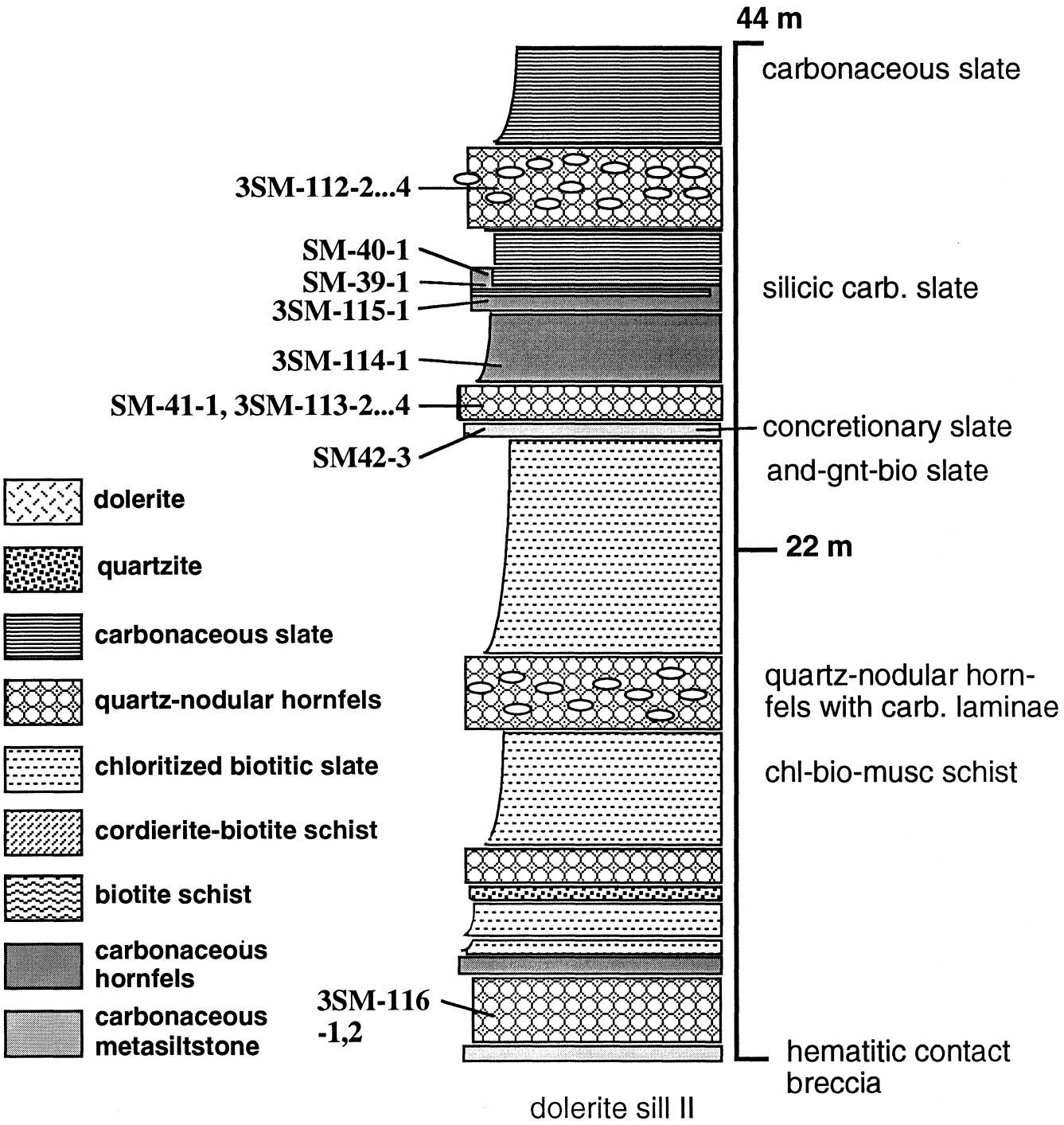
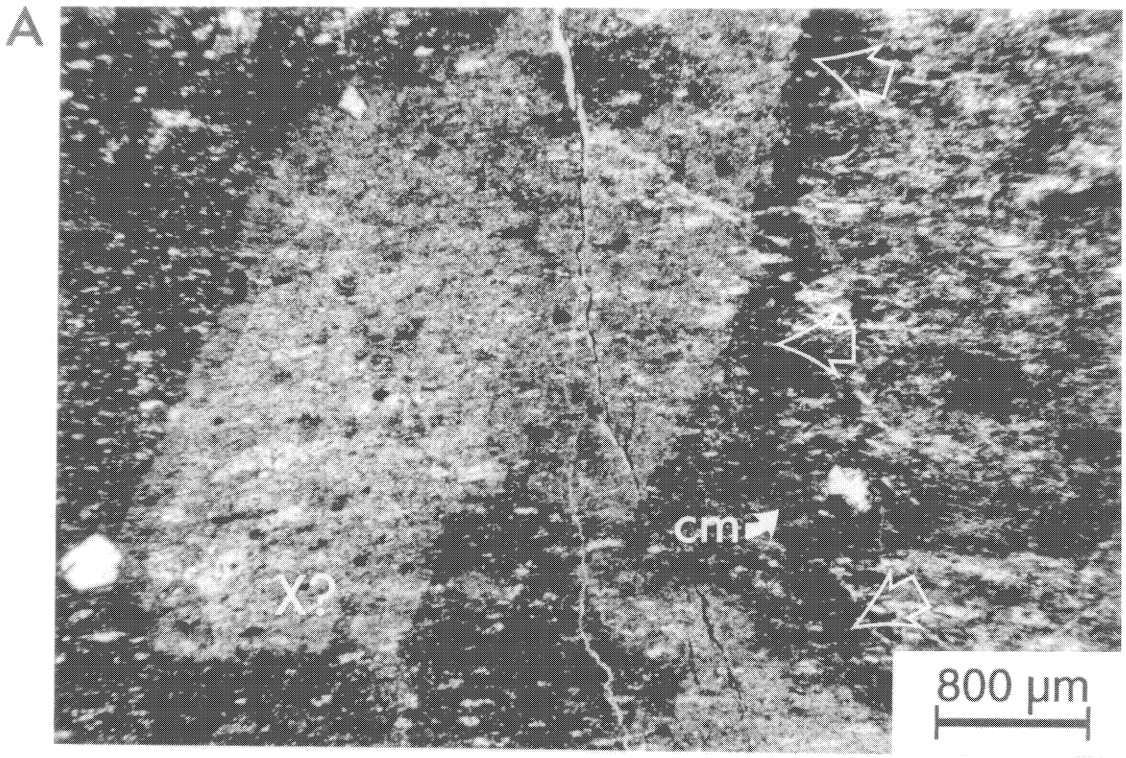
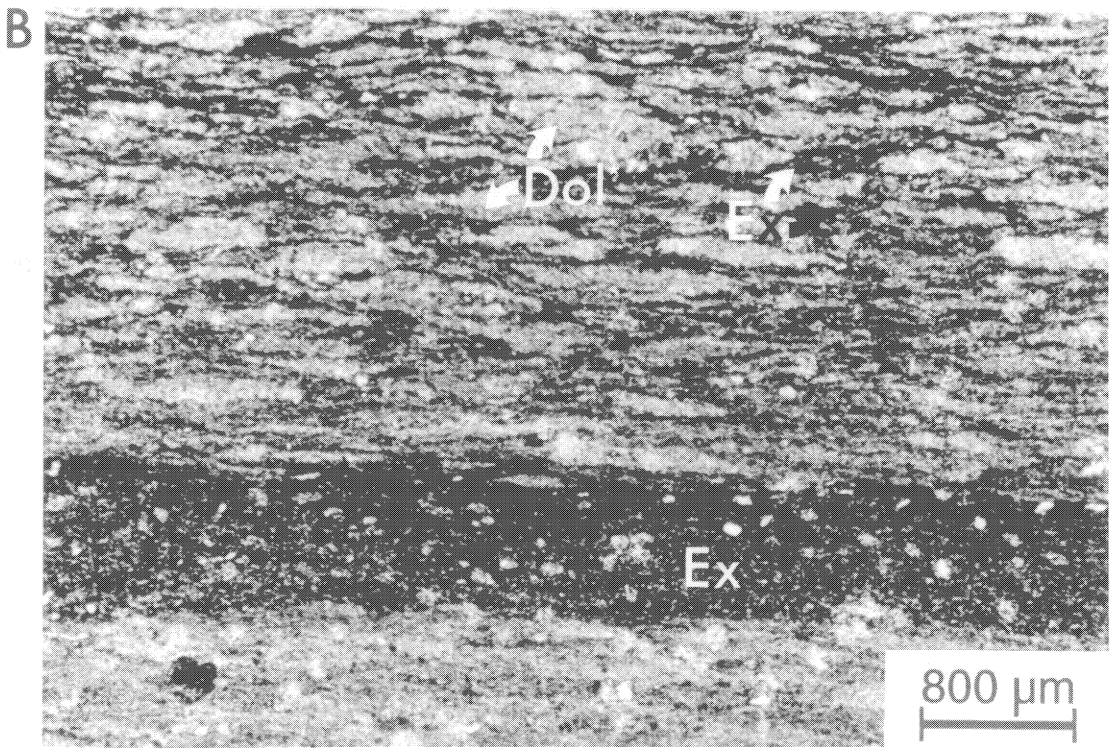
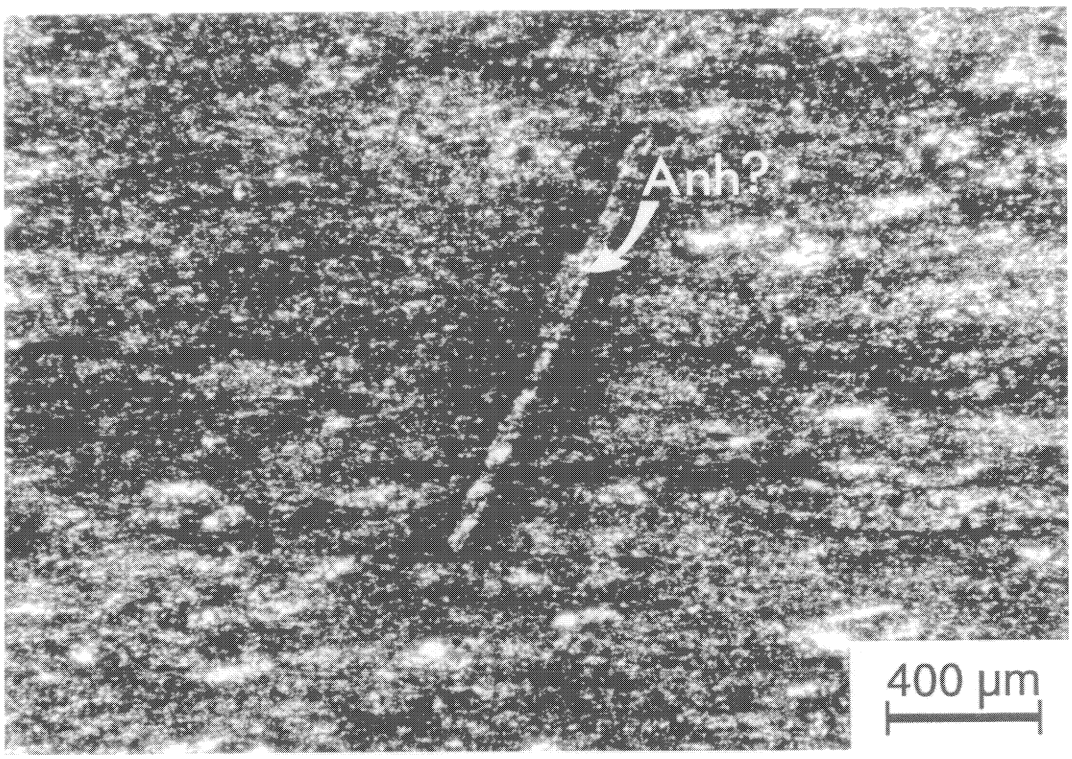


Figure 2

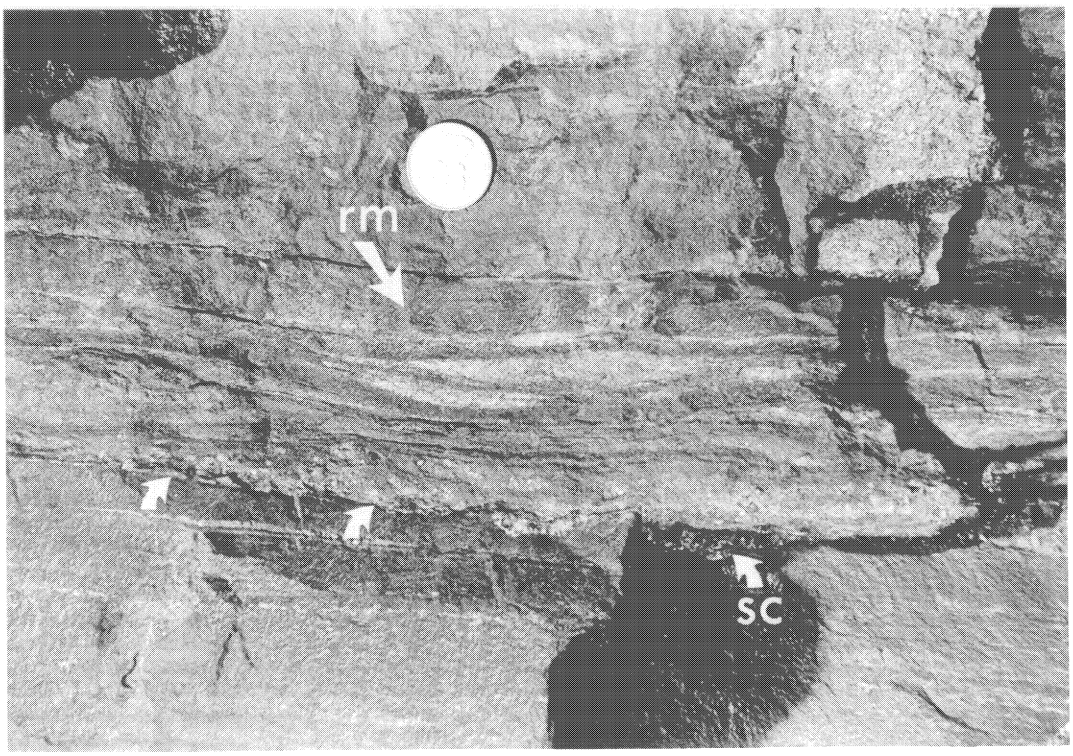

A
cm
X?
800 µm


B
Dol
Ex
Ex
800 µm

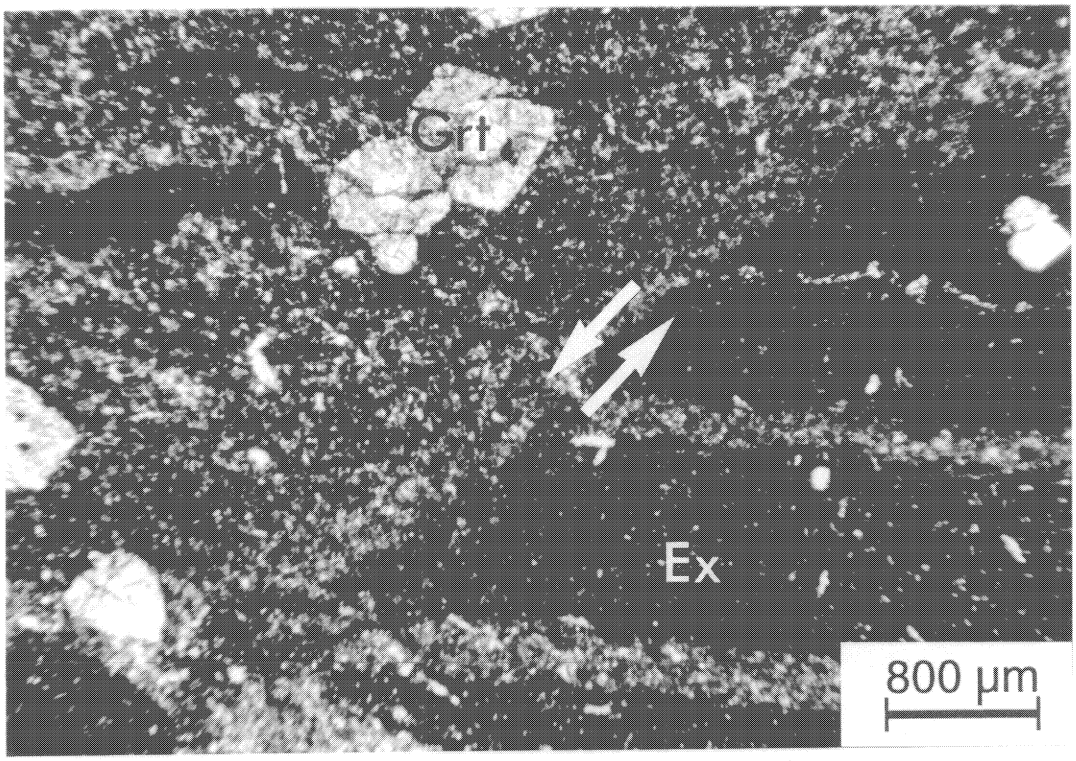
C



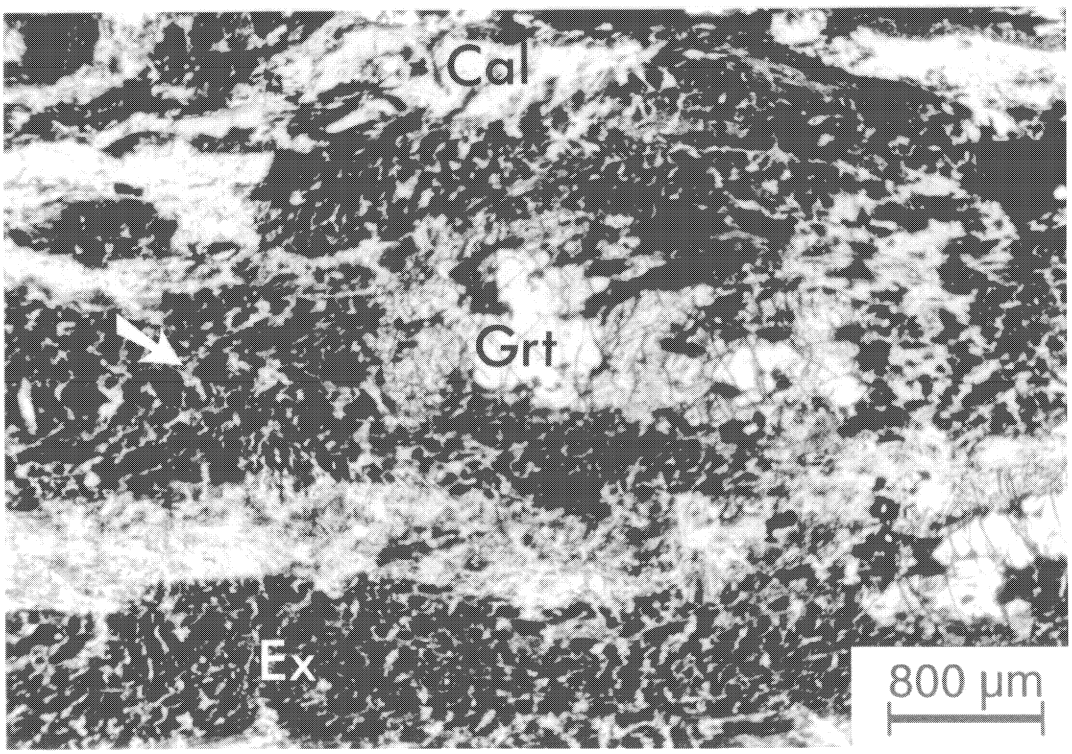
D



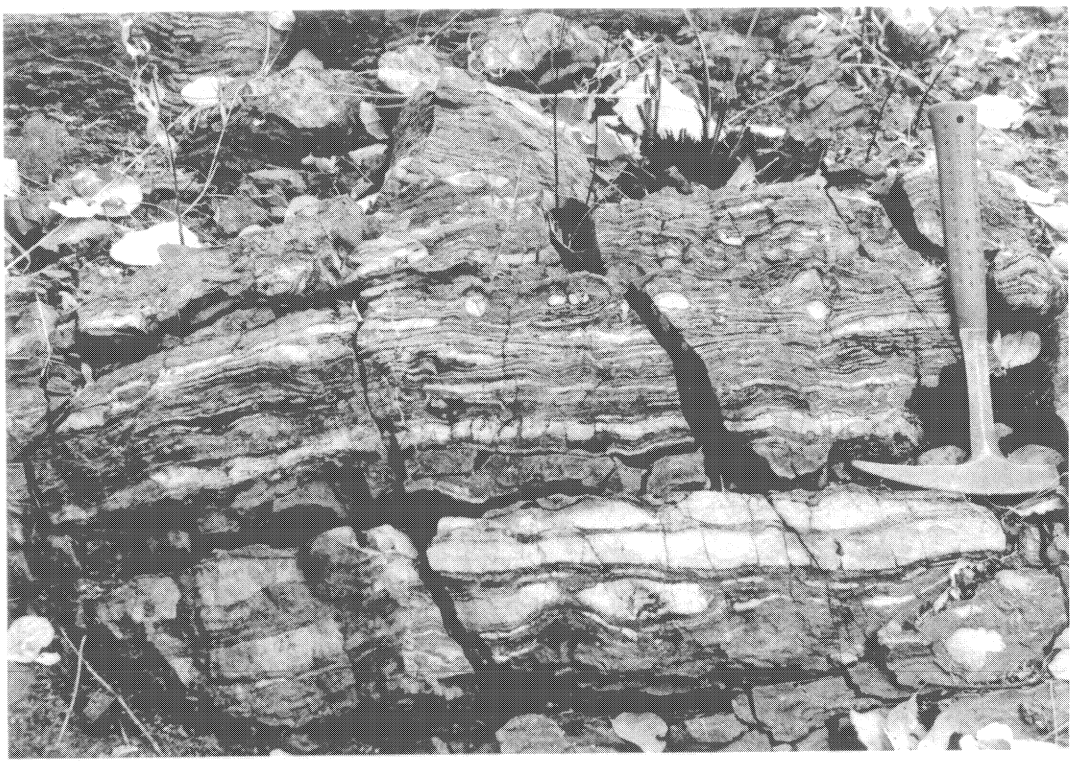
E



F



A



B

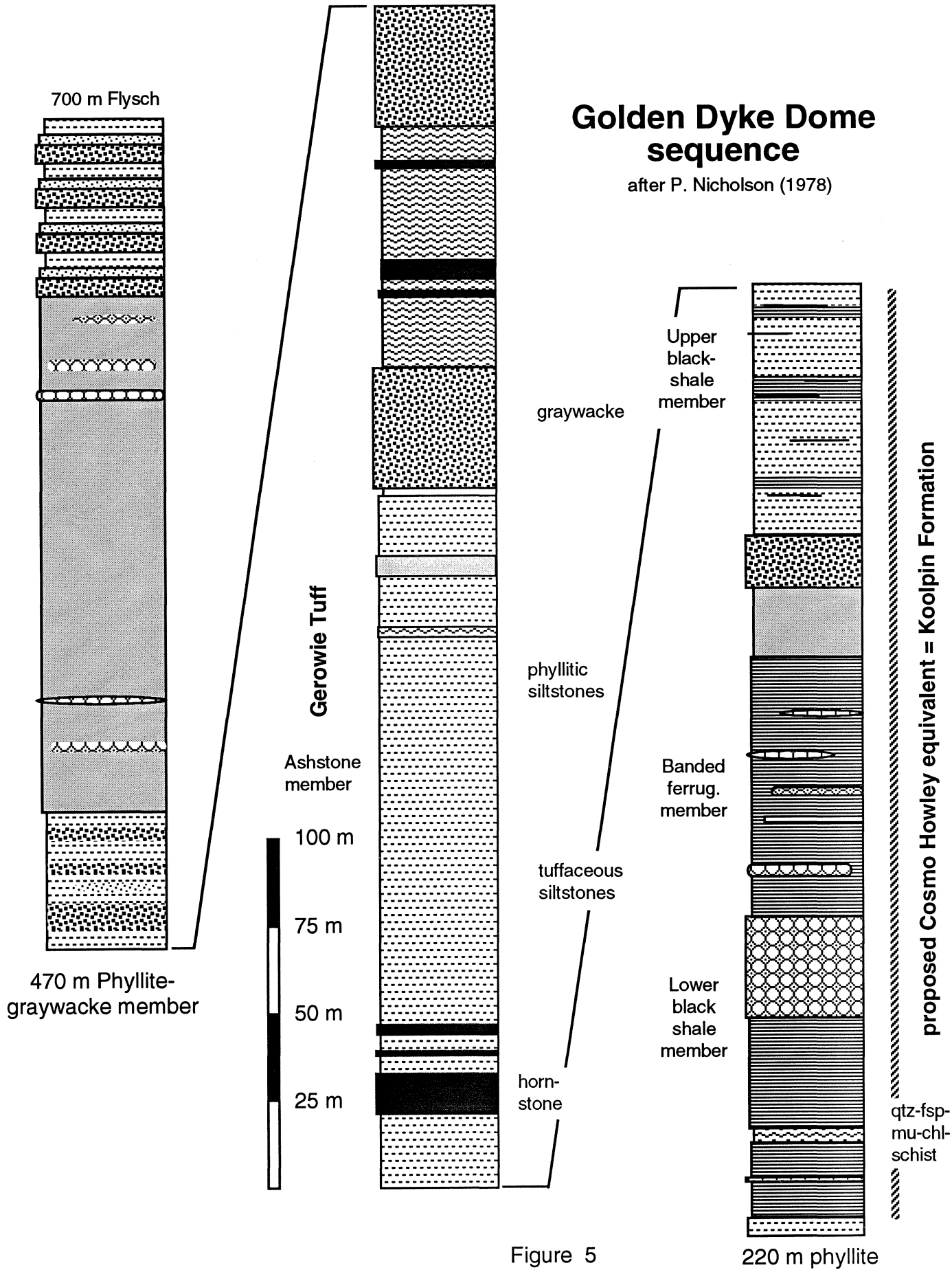

Golden Dyke Dome sequence
after P. Nicholson (1978)
700 m Flysch
470 m Phyllite-graywacke member
Gerowie Tuff
Ashstone member
100 m
75 m
50 m
25 m
graywacke
phyllitic siltstones
tuffaceous siltstones
horn-stone
Upper black-shale member
Banded ferrug. member
Lower black shale member
220 m phyllite
proposed Cosmo Howley equivalent = Koolpin Formation
qtz-fsp-mu-chl-schist

Figure 5

Moinik Formation,
Lake Magadi, Kenya

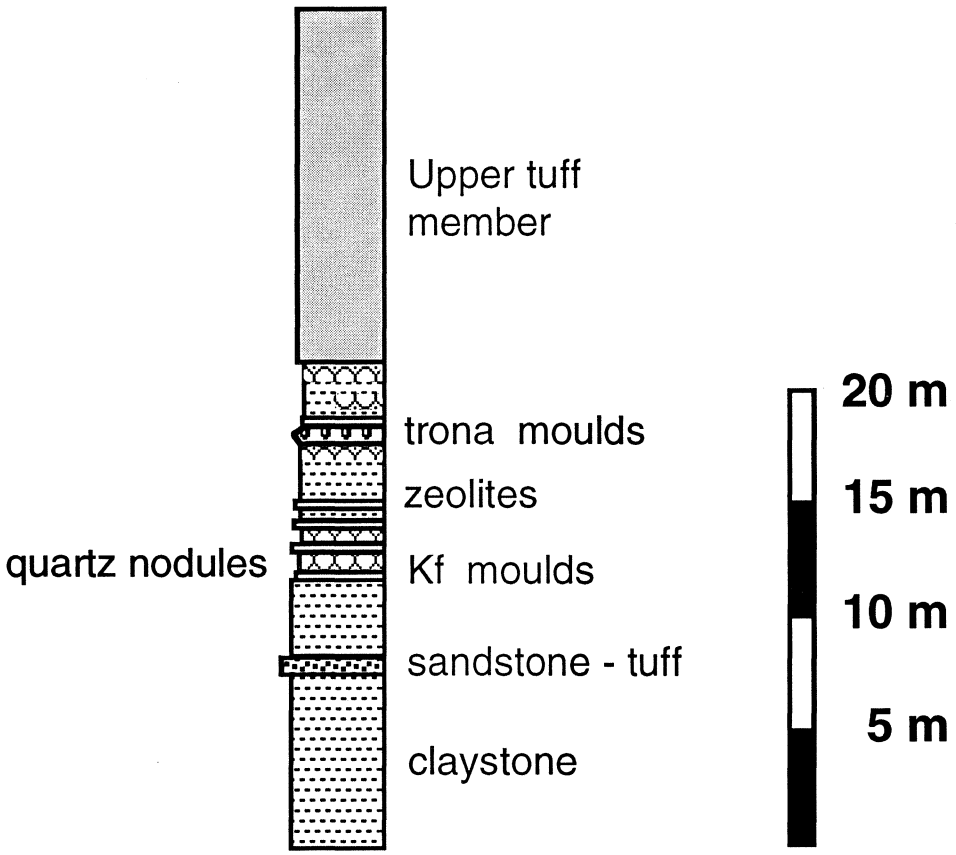


Figure 6

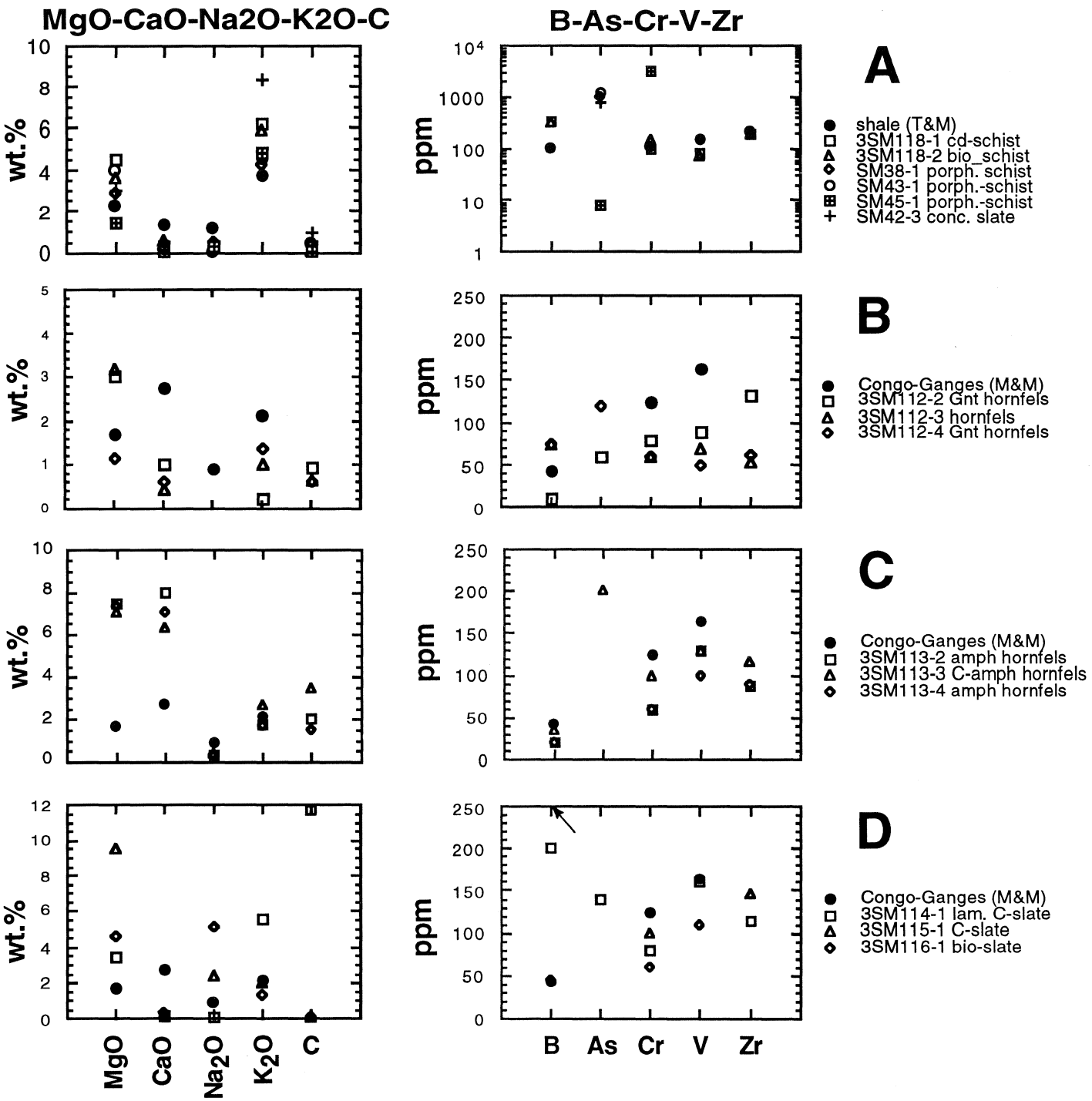

MgO-CaO-Na2O-K2O-C
B-As-Cr-V-Zr
wt.%
ppm
A
shale (T&M)
3SM118-1 cd-schist
3SM118-2 bio_schist
SM38-1 porph. schist
SM43-1 porph.-schist
SM45-1 porph.-schist
SM42-3 conc. slate
B
Congo-Ganges (M&M)
3SM112-2 Gnt hornfels
3SM112-3 hornfels
3SM112-4 Gnt hornfels
C
Congo-Ganges (M&M)
3SM113-2 amph hornfels
3SM113-3 C-amph hornfels
3SM113-4 amph hornfels
D
Congo-Ganges (M&M)
3SM114-1 lam. C-slate
3SM115-1 C-slate
3SM116-1 bio-slate
MgO
CaO
Na2O
K2O
C
B
As
Cr
V
Zr

Figure 7

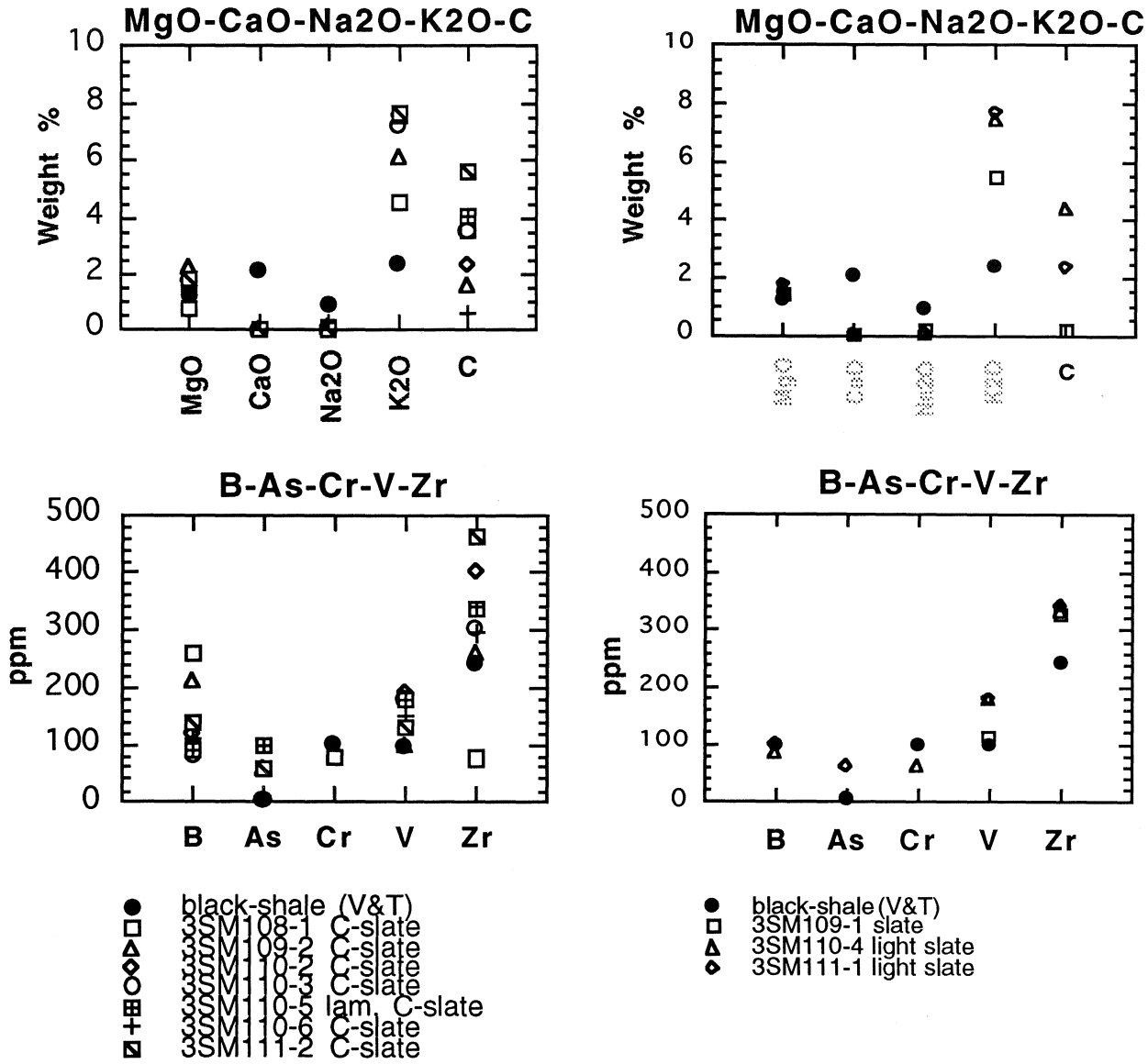

MgO-CaO-Na2O-K2O-C
Weight %
10
8
6
4
2
0
MgO
CaO
Na2O
K2O
C
B-As-Cr-V-Zr
ppm
500
400
300
200
100
0
B
As
Cr
V
Zr
black-shale (V&T)
3SM108-1 C-slate
3SM109-2 C-slate
3SM110-2 C-slate
3SM110-3 C-slate
3SM110-5 lam. C-slate
3SM110-6 C-slate
3SM111-2 C-slate
black-shale (V&T)
3SM109-1 slate
3SM110-4 light slate
3SM111-1 light slate

Figure 8

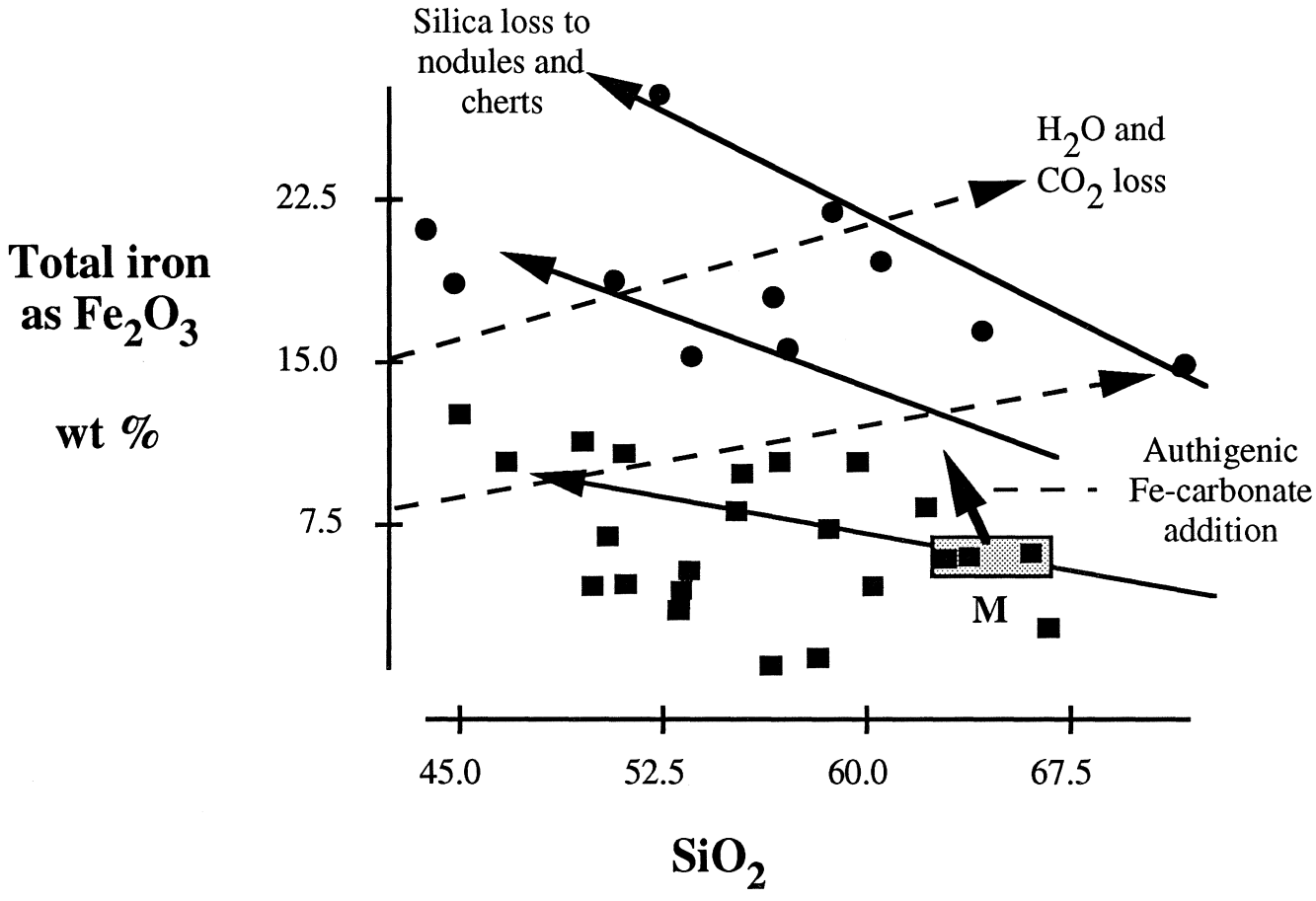

Silica loss to nodules and cherts
H_2O and CO_2 loss
Authigenic Fe-carbonate addition
M
22.5
15.0
7.5
45.0
52.5
60.0
67.5
Total iron as Fe_2O_3
wt %
SiO_2

Figure 9

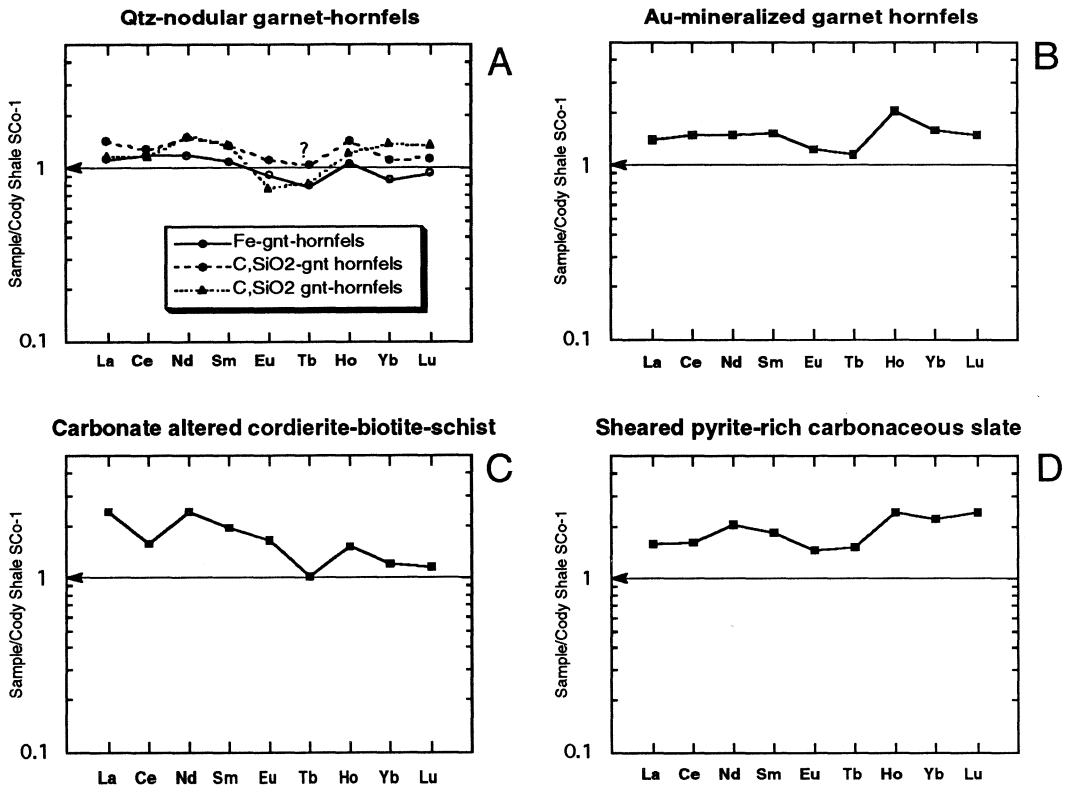

Qtz-nodular garnet-hornfels
A
Sample/Cody Shale SCo-1
1
0.1
La Ce Nd Sm Eu Tb Ho Yb Lu
?
Fe-gnt-hornfels
C,SiO2-gnt hornfels
C,SiO2 gnt-hornfels
Au-mineralized garnet hornfels
B
Carbonate altered cordierite-biotite-schist
C
Sheared pyrite-rich carbonaceous slate
D

Figure 10

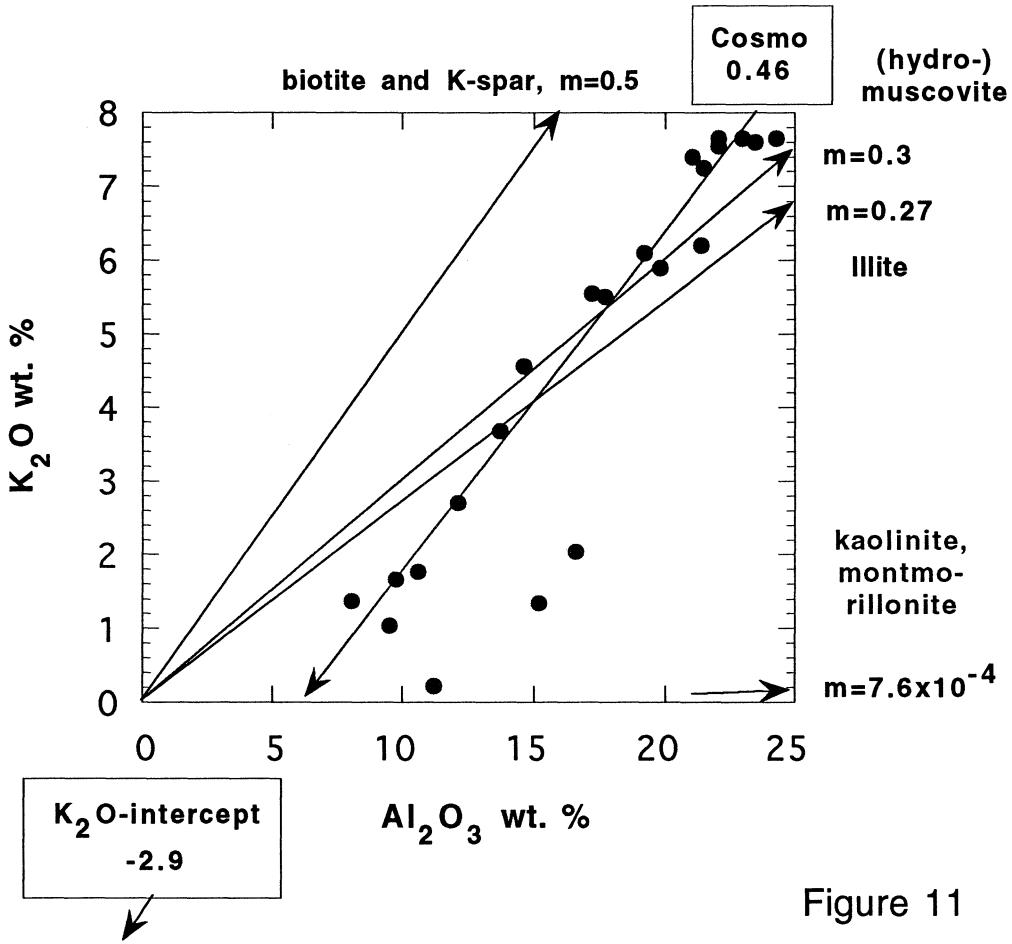

Cosmo
0.46
(hydro-)
muscovite
biotite and K-spar, m=0.5
m=0.3
m=0.27
Illite
kaolinite,
montmo-
rillonite
m=7.6x10-4
K2O wt. %
Al2O3 wt. %
K2O-intercept
-2.9
0
1
2
3
4
5
6
7
8
0
5
10
15
20
25

Figure 11

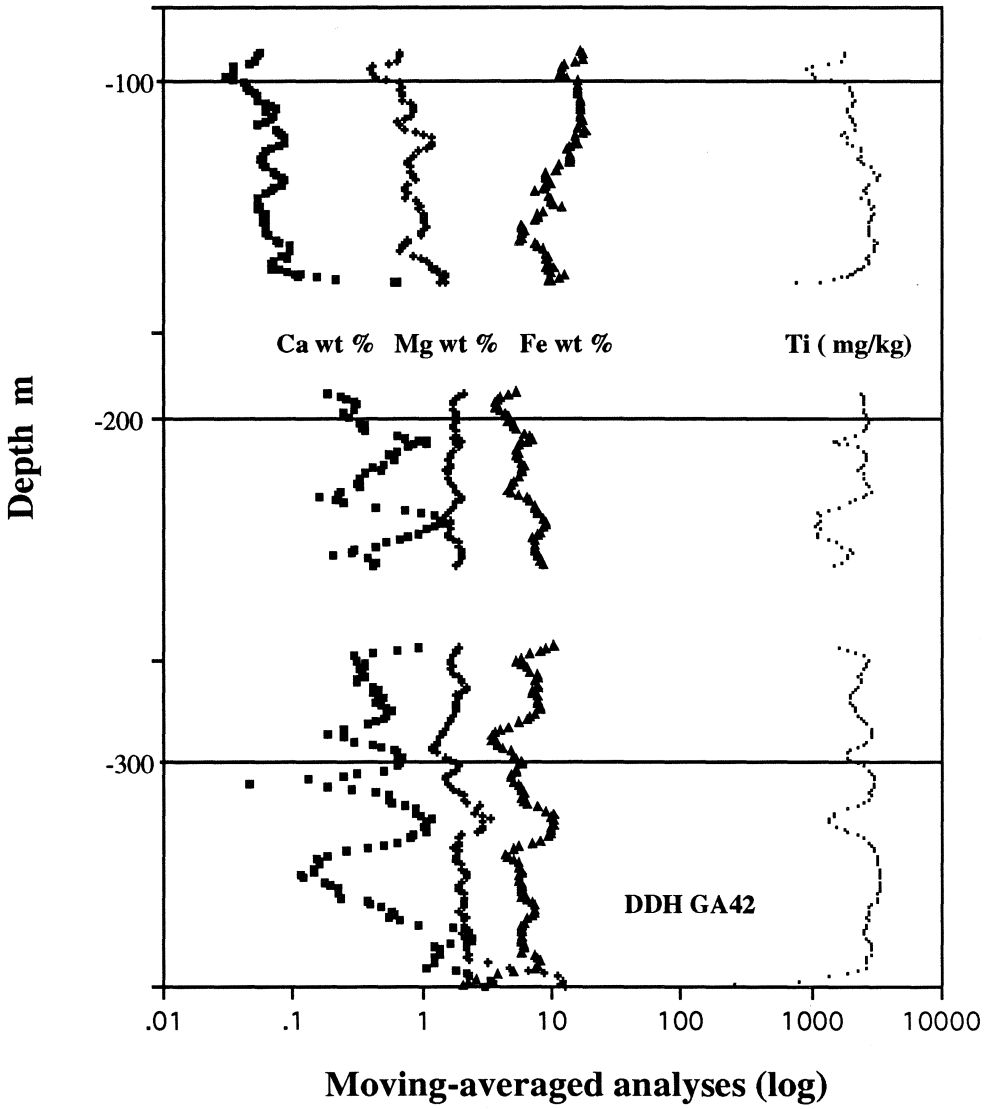

-100
-200
-300
Depth m
Ca wt %
Mg wt %
Fe wt %
Ti (mg/kg)
DDH GA42
.01
.1
1
10
100
1000
10000
Moving-averaged analyses (log)

Figure 12

B Tables

Sample	La	Ce	Nd	Sm	Eu	Tb	Ho	Yb	Lu	Total REE
Fe-rich gnt hornfels	33.30	70.80	29.30	5.11	1.06	0.67	0.70	1.61	0.27	142.82
C-rich gnt hornfels	41.70	76.30	36.90	6.34	1.29	0.88	0.94	2.09	0.33	166.77
C-rich gnt hornfels	34.80	69.30	37.30	6.40	0.89	0.70	0.80	2.57	0.39	153.15
Au-rich gnt hornfels	42.00	90.50	37.20	7.17	1.44	0.98	1.34	2.98	0.43	184.40
Sheared py-rich C-slate	47.20	97.80	50.70	8.67	1.70	1.31	1.61	4.20	0.70	112.01
Cd-bio-cc schist	71.50	96.40	60.10	9.17	1.91	0.88	1.00	2.30	0.34	243.60
Cody Shale (SCo-1)	29.60	60.00	24.70	4.71	1.16		0.66	1.88	0.29	123.00

Table 1: Rare-earth element data for selected samples, Cosmo Howley open pits. Units are ppm. Py = pyrite, cd = cordierite, bio = biotite, cc = calcite.

Table 2: Next two pages; whole rock combined XRF-ICP analysis of selected samples of metasediments at the Cosmo Howley Deposit. Major elements (XRF) in wt.%; trace elements (ICP) in ppm, detection limits: B, As, Cr = 50, V = 20, Zr = 10; n.a. = not analyzed. Analysis by Classic Laboratories Ltd., Wangera, Western Australia.

	Reference Compositions				Carbonaceous Slate 1				
	Black Shale V&T	Shale T&M	Shale	Mud M&M	3SM 108-1	3SM 109-2	3SM 110-2	3SM 110-3	3SM 110-5
SiO_2	66.60	62.80	61.80	62.50	66.70	60.30	56.50	53.40	51.20
TiO_2	0.30	1.00	0.89	1.30	0.57	0.047	0.57	0.45	0.48
Al_2O_3	15.00	18.90	15.00	20.50	14.60	19.20	23.50	21.50	22.10
Fe_2O_3	2.70	6.50	6.50	7.80	2.70	4.62	1.05	5.20	4.72
MnO	0.019	0.11	0.0393	0.17	0.01	0.03	0.01	0.01	0.01
MgO	1.20	2.20	1.50	1.70	0.79	2.26	1.73	1.59	1.70
CaO	2.10	1.30	4.75	2.75	0.01	0.01	0.02	0.01	0.01
Na_2O	0.90	1.20	0.60	0.90	0.10	0.09	0.07	0.07	0.06
K_2O	2.40	3.70	2.45	2.15	4.56	6.10	7.60	7.25	7.55
P_2O_5	n.a.	0.16	0.16	n.a.	0.02	0.03	0.04	0.08	0.04
S	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
C	n.a.	n.a.	1.14	n.a.	3.59	1.60	2.33	3.54	4.04
L.O.I.	n.a.	n.a.	n.a.	n.a.	8.5	6.60	7.40	10.40	10.90
Total	91.22	97.87	94.83	99.77	98.56	99.28	98.49	99.95	98.77
B	100	100	200	43	260	210	120	80	90
As	6	<	<	<	<	60	<	<	100
Cr	100	110	71	123	80	<	<	<	<
V	99	150	64.6	163	180	100	190	180	180
Zr	243	210	18	<	77	260	402	305	339

	Carbonaceous Slate		Biotitic Slate		Light Slate			Garnet Hornfels	
	3SM 110-6	3SM 111-2	SM 42-3	3SM 116-1	3SM 109-1	3SM 110-4	3SM 111-1	3SM 112-2	3SM 112-4
SiO_2	58.10	49.90	50.50	51.10	55.40	53.20	53.10	52.40	60.60
TiO_2	0.51	0.60	0.70	0.49	0.50	0.47	0.48	0.55	0.26
Al_2O_3	24.30	22.10	23.00	16.60	17.70	21.08	23	11.2	8.05
Fe_2O_3	1.47	4.76	6.9	10.80	9.90	4.58	3.46	27.20	19.60
MnO	0.01	0.01	0.08	0.06	0.02	0.01	0.01	1.05	0.79
MgO	1.49	1.85	3.00	9.50	1.40	1.66	1.77	3.04	1.17
CaO	0.02	0.03	0.17	0.28	0.01	0.01	0.02	1.03	0.62
Na_2O	0.1	0.06	0.28	2.38	0.18	0.06	0.07	<	<
K_2O	7.65	7.65	8.30	2.04	5.50	7.40	7.65	0.22	1.37
P_2O_5	0.09	0.05	0.12	0.08	<	0.05	0.06	0.09	0.09
S	n.a.	n.a.	<	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
C	0.61	5.58	0.95	0.11	0.16	4.38	2.35	0.95	0.64
L.O.I.	5.75	11.6	5.5	6.30	9.05	10.80	9.15	3.48	7.50
Total	99.49	98.61	99.5	99.63	99.82	99.32	98.77	100.26	100.05
B	110	140	300	45	<	85	100	10	75
As	<	60	750	<	<	<	60	60	120
Cr	<	<	50	60	<	60	<	80	60
V	150	130	50	110	110	180	180	90	50
Zr	295	463	400	288	327	333	342	132	62

	Amphibole Hornfels			Partly Carbonaceous Hornfels					Hornfels
	3SM 113-2	3SM 113-3	3SM 113-4	3SM 114-1	3SM 115-1	3SM 116-2	SM 39-1	SM 40-1	3SM 112-3
SiO_2	43.80	44.90	50.70	53.50	46.80	62.90	62.10	66.10	64.30
TiO_2	0.44	0.59	0.39	0.51	0.66	0.33	0.53	0.56	0.17
Al_2O_3	10.60	12.10	9.75	13.70	17.20	15.20	14.40	15.40	9.50
Fe_2O_3	21.00	18.70	18.7	15.10	10.50	5.90	8.30	6.30	16.40
MnO	0.59	0.51	0.55	0.69	0.16	0.04	0.07	0.02	0.45
MgO	7.45	7.05	7.3	3.86	3.48	4.62	2.10	1.10	3.20
CaO	7.95	6.3	7.05	0.41	0.17	0.36	0.12	0.03	0.43
Na_2O	0.34	0.37	0.33	<	0.10	5.15	0.13	0.20	<
K_2O	1.77	2.70	1.67	3.68	5.55	1.34	3.90	4.88	1.04
P_2O_5	0.05	0.06	0.06	0.06	0.6	0.07	0.1	0.02	0.07
S	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	<	<	n.a.
C	2.04	3.46	1.51	5.47	11.70	0.14	2.35	1.85	0.66
L.O.I.	6.55	6.65	4.1	7.75	15.00	3.20	6.65	4.64	5.20
Total	100.59	99.93	100.60	99.26	100.22	99.11	100.75	101.10	100.67
B	20	35	20	200	1400	40	300	200	75
As	<	200	<	140	<	<	<	<	<
Cr	60	100	60	80	100	60	300	150	60
V	130	130	100	160	280	80	100	120	70
Zr	87	116	89	114	146	160	150	100	53

	Hornfels				Porphyroblastic Schist				Bio-Schist
	SM 2-2	SM 3-1	SM 8-2	SM 41-1	3SM 118-1	SM 38-1	SM 43-1	SM 45-1	3SM 118-2
SiO_2	56.70	57.10	58.80	71.50	49.40	58.50	59.60	63.80	55.10
TiO_2	0.65	0.53	0.27	0.27	0.84	0.33	0.55	0.63	0.86
Al_2O_3	16.60	12.20	6.70	6.55	21.40	18.90	13.80	16.10	19.80
Fe_2O_3	10.50	15.40	22.00	14.70	11.40	7.25	10.50	6.10	8.30
MnO	0.76	4.00	2.84	0.02	0.08	0.03	0.23	0.03	0.06
MgO	4.00	3.22	3.00	0.70	4.46	2.84	3.94	1.45	3.58
CaO	0.95	1.78	1.31	0.07	0.32	0.10	0.33	0.09	0.52
Na_2O	3.42	0.07	0.04	0.03	0.19	0.43	0.05	0.29	0.30
K_2O	3.82	2.26	0.53	1.88	6.20	4.20	4.54	4.82	5.90
P_2O_5	0.12	0.10	<	0.06	0.18	0.05	0.22	0.08	0.35
S	0.95	1.05	0.70	<	n.a.	<	1.05	<	n.a.
C	0.2	0.35	2.5	0.3	0.27	0.35	0.45	0.05	0.02
L.O.I.	2.26	3.48	4.70	3.42	6.45	3.86	4.40	4.10	4.62
Total	100.93	101.54	98.69	99.50	100.92	96.84	99.66	97.54	99.39
B	100	600	100	100	340	4000	200	200	320
As	<	<	1050	1150	<	1000	1150	3200	<
Cr	150	150	150	200	100	100	350	200	140
V	40	40	40	40	80	<	50	30	70
Zr	150	100	50	50	18 n.a.	250	100	150	183