UNIVERSITY OF ADELAIDE LIBRARY
28.4.98

VISOR++:
A SOFTWARE VISUALISATION TOOL FOR TASK-PARALLEL OBJECT-ORIENTED PROGRAMS

Hendra Widjaja

A THESIS SUBMITTED FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE
IN THE DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ADELAIDE

March 1998

visor **n** 1. (*hist*) Movable part of a helmet, covering the face. 2. Peak of a cap. 3. (**'sun-**)~, oblong sheet of dark-tinted glass hinged at the top of a windscreen in a car to lessen the glare of bright sunshine.

(*Oxford Advanced Learner's Dictionary of Current English, by A.S. Hornby, Oxford University Press, 3rd edition, 1980, page 959.*)

VISOR Acronym for **Visual Instrument and Sensory Organ Replacement**. A remarkable piece of bioelectronic engineering that allowed Geordi La Forge to see, despite the fact that he was born blind. A slim device worn over the face like a pair of sunglasses, the Visor permitted vision in not only visible light, but across spectrum, including infrared and radio waves.

(*The Star Trek Encyclopedia, A Reference Guide to the Future, by M. Okuda, D. Okuda and D. Mirek, Pocket Books, New York, 1994, page 368.*)

Abstract

Applying software visualisation to task-parallel object-oriented programs poses interesting questions. The reason for this is that, typically, such programs exhibit complex behaviour as a result of the complex interaction among the program entities. Such interaction is caused, in part, by concurrency and distribution.

With the exception of a limited number of tools, many existing tools only focus on a narrow selection of language features for visualisation. However, to enable users to form a deep understanding, and subsequently fine-tune a program, a wide selection of such features is necessary for visualisation. Furthermore, multiple views depicting the program from multiple angles are also necessary.

This thesis describes Visor++, a tool for visualising programs written in CC++, a task-parallel, object-oriented language derived from C++. Visor++ provides a framework of visualising task-parallel object-oriented programs in the absence of language support for visualisation. In other words, Visor++ provides support for the visualisation of programs written in languages which are not "visualisation-conscious"; CC++ is one such language.

This thesis describes the techniques developed to enable the visualisation of task-parallel object-oriented programs by using a wide selection of language features. The effectiveness of this approach is testified by the experimentation with the tool. The design and experimentation with Visor++ are all described in this thesis.

Although the framework of Visor++ is implemented on specific platforms, it can also be applied to other similar systems.

Declaration

This is to certify that this thesis contains no material which has previously been accepted for the award of any degree or diploma in any university or other tertiary institution. To the best of my knowledge and belief, it contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

If this thesis is accepted for the award of the degree, permission is granted for it to be made available for loan and photocopying.

Hendra Widjaja
March 1998

Acknowledgements

First of all, I would like to thank my supervisor, Dr Michael J. Oudshoorn, for his advice, encouragement, and superb guidance during my candidature as a Master's student. He is the one who introduced me to the exciting field of software visualisation. I am also much indebted to Dr Jiannong Cao, who acted as co-supervisor during the first semester of 1995, particularly when Dr Oudshoorn was away from March to June 1995. His advice and guidance were of utmost importance during the infancy of the project.

In 1997, I presented a paper in San Jose, USA. Once again, I am much indebted to my supervisor, Dr Oudshoorn, for his tireless efforts in obtaining financial support for the trip. He is indeed a super "Visor". I also wish to thank Garuda Indonesia for providing special arrangements for the trip.

I owe special thanks to Prof. Peter Eades and Dr. Kang Zhang for reading the the thesis. Their comments are particularly insightful and invaluable, especially for further enhancements of the work.

Acknowledgment and thanks also go to the staff and members of the Department of Computer Science for all their help and support, particularly to Matthew, Stuart, Sam and Heath for their superb technical assistance. Special thanks to the DHPC (Distributed High-Performance Computing) group at the department for letting me print portions of the thesis by using their colour printer. Other members of the staff, the postgraduate students and visiting speakers have also contributed in many ways. Many thanks also to my office colleague, Lin Huang, for teaching me how to make dumplings.

I also wish to acknowledge the technical help I received that has made my work possible. Special thanks to the CC++ language developers at California Institute of Technology, who made their system accessible, and provided much technical support through electronic mail by answering my questions, even the stupid ones. I would also like to express my special thanks to Professor John Stasko at Georgia Institute of Technology, who made his POLKA system publicly available, upon which my work is based.

Thank you, too, to all my friends, both at home in Indonesia, and here in Australia. They have made my stay in Adelaide more enjoyable. Many thanks, too, to the staff at CISSA (Council for International Students of South Australia), who have provided international students such as myself a chance to glimpse into the lives of Australians and the people of other cultures. I would also like to extend my thanks to AusAID for providing the Australian Development Cooperation Scholarship (ADCOS), without which my work and stay in Australia would not have been possible.

Finally, I would like to express my deepest gratitude to my parents and family, who have constantly supported and prayed for me from afar.

Contents

Abstract iii

Declaration iv

Acknowledgements v

1 Introduction 1

1.1 Motivation 5

1.2 Experimentation 8

1.3 Terminology 8

1.4 Thesis Structure 9

2 Software Visualisation 10

2.1 Aspects of Software Visualisation 10

2.1.1 Ideals 11

2.1.2 Purposes 12

2.1.2.1 Understanding 13

2.1.2.2 Debugging 13

2.1.2.3 Performance Analysis 15

2.1.3 Mechanisms 17

2.1.3.1 Data Collection 17

2.1.3.2 Visualisation 18

2.2 Visualisation Systems 20

2.2.1 Sequential Systems 21

2.2.2 Concurrent Systems 22
2.2.3 Concurrent Object-Oriented Systems 26
2.2.3.1 Visualisation of μC++ programs 26
2.2.3.2 Visualisation of pC++ programs 28
2.2.3.3 Visualisation of LAMINA programs 29
2.2.3.4 Visualisation of PARC++ programs 30
2.2.3.5 Observations 31
2.3 Summary 31

3 Overview of the CC++ Language 34
3.1 The CC++ Language 34
3.1.1 Concurrency 35
3.1.1.1 Synchronous threads 35
3.1.1.2 Asynchronous threads 36
3.1.2 Synchronisation and Determinism 37
3.1.2.1 Synchronisation 37
3.1.2.2 Atomic Functions 37
3.1.3 Locality 39
3.1.3.1 Processor Objects 39
3.1.3.2 Remote Procedure Calls 40
3.2 Visualisation Support 43

4 Visor++ 45
4.1 General Framework 46
4.2 Program Instrumentation Subsystem 48
4.2.1 Program Entities 49
4.2.2 Program Static Analysis 50
4.2.3 Program Instrumentation 52
4.2.3.1 Events 53
4.2.3.2 Location-ID 55
4.2.3.3 Instrumenting Functions 57

4.2.3.4 Instrumenting Destructors 62
4.2.3.5 Instrumenting Processor Object Classes 66
4.2.3.6 Instrumenting Synchronous Threads. 70
4.2.3.7 Instrumenting Asynchronous Threads. 73
4.2.4 Observations 74
4.3 Event Collection Subsystem 78
4.3.1 Establishing the Monitoring Environment 78
4.3.2 Collecting Traces 80
4.3.3 Consolidating traces 83
4.3.4 Observations 85
4.4 Event Visualisation Subsystem 87
4.4.1 POLKA 87
4.4.2 Architecture and Implementation 88
4.4.3 The Views 92
4.4.3.1 Design Considerations 93
4.4.3.2 Static Views 97
4.4.3.3 Auxiliary Views 101
4.4.3.4 Dynamic Views 102
4.4.4 Observations 119
4.5 Summary 119
4.6 Applicability to Other Systems 123

5 Using Visor++ 125

5.1 Experiments 125
5.1.1 A Simple Example 126
5.1.2 Distributed Merge-Sort 127
5.1.3 Concurrent String Search 131
5.1.3.1 Implementation-1 131
5.1.3.2 Implementation-2 131
5.1.3.3 Implementation-3 135
5.1.4 An Electronic Transaction System 136

5.1.4.1 Implementation-1 138
5.1.4.2 Implementation-2 140
5.1.4.3 Implementation-3 146
5.1.4.4 Implementation-4 148
5.2 Discussion 151
5.2.1 Merits 151
5.2.2 Limitations on Visor++ Usage 152

6 Conclusions and Future Work **155**
6.1 Summary 155
6.2 Conclusions 158
6.3 Future Work 158

A An Instrumentation Example **161**
A.1 The Original Code 161
A.2 The Instrumented Code 163

B Transaction Subsystem Code **167**
B.1 The Transaction Requests 167
B.2 The Stock and Customer Databases 168
B.3 The Business Logic Unit 170
B.4 The Transaction Servers 170
B.5 The Transaction Resolution Subsystem 171

List of Tables

5.1 Timing information from the merge-sort program. 129

5.2 Timing information from the parallel text-searching programs. 135

5.3 Timing information from Implementation-3 and Implementation-4. . . . 149

List of Figures

1.1 *Data visualisation of air flow near the airfoils of an aircraft.* 2

1.2 *Constructing a procedure in Hyperpascal.* 3

1.3 *The* **Processor Communication View** *in ParaGraph.* 4

1.4 *Algorithm animation of the Tower of Hanoi, using POLKA.* 5

1.5 *Mental models in software or program comprehension.* 7

2.1 *Aspects of software visualisation.* 10

2.2 *The* **Classy** *tool in the TAU visualisation toolset.* 14

2.3 *The* **Coarse Grained View** *in TPM.* 14

2.4 *The control panel of the parallel debugger* **Breezy** *in TAU.* 15

2.5 *Visualising data structures in* **Breezy**. 16

2.6 *The* **Utilisation View** *in ParaGraph.* 16

2.7 *Transformations to produce software visualisation.* 17

2.8 *The different views in visualisation tools.* 20

2.9 *The* **Spacetime Diagram** *in ParaGraph.* 23

2.10 *Statistics of communication traffic in ParaGraph.* 24

2.11 *The* **History View** *and the* **Mutex View** *in Gthreads.* 24

2.12 *Trace visualisation in MVD (POET).* 27

3.1 *Usage of the* **par** *construct.* 35

3.2 *Implicit synchronisation barrier at the end of a* **par** *or* **parfor** *block.* . 36

3.3 *Usage of the* **parfor** *construct.* 36

3.4 *Usage of the* **spawn** *construct.* 37

3.5 *Usage of the* **sync** *construct.* 38

3.6 *Usage of the* **atomic** *construct.* 38

3.7 *Mapping of threads and processor objects in CC++.* 39

3.8 *Implementation of a processor object class.* 40

3.9 *Usage of global pointers.* 41

3.10 *Data communication and RPC between two processor objects.* 42

3.11 *A transfer function.* 43

4.1 *General framework of Visor++.* 46

4.2 *Information on functions in the static repository.* 50

4.3 *Information on classes in the static repository.* 51

4.4 *Ill-partitioned, but valid program.* 52

4.5 *Automatically obtaining a unique identifier for a processor object.* . . . 58

4.6 *A light-weight atomic function to obtain unique thread identifiers.* . . . 59

4.7 *Program trace structure.* 59

4.8 *Example of function instrumentation.* 61

4.9 *A function can have many exit paths.* 62

4.10 *Profiler class in for the instrumentation subsystem.* 63

4.11 *A function instrumented with an* **EntityProfiler** *object.* 64

4.12 *Example of an altered function call.* 65

4.13 *Constructor and destructor called from different threads.* 66

4.14 *Instrumentation of a class.* 67

4.15 *Changing the call to an object destructor.* 68

4.16 *Instrumenting a member function of a processor object class.* 69

4.17 *Marking the start and end of a RPC.* 70

4.18 *The generation of events for a RPC.* 71

4.19 *A drawback in RPC instrumentation.* 71

4.20 *Simple re-arrangment of source-code for RPC instrumentation.* 72

4.21 *Instrumentation of a* **parfor** *or* **par** *block.* 73

4.22 *Instrumenting a* **parfor** *block and its threads.* 74

4.23 *Instrumenting a* **par** *block and its threads.* 75

4.24 *Instrumenting a* **spawn** *block.* 76

4.25 *An unfortunate consequence of introducing a new scope.* 76
4.26 *Simple re-arrangement of source code as a way out.* 77
4.27 *Allocation of monitoring processes.* 79
4.28 *Collecting program traces.* 81
4.29 *Consolidating program traces.* 84
4.30 *Causality violations in event ordering.* 86
4.31 *The corrected DAG reflecting the "happened-before" relationships.* . . . 86
4.32 *Classes in POLKA, and their* **has-a** *relationships.* 88
4.33 *Example program of the animation of a circle.* 89
4.34 *The event visualisation subsystem.* 91
4.35 *Visor++ in execution.* 94
4.36 *Relationships among views.* 95
4.37 *The* **Source-Code View**. 98
4.38 *The* **Class Hierarchy View**. 99
4.39 *The* **Class Information View**. 100
4.40 *Information on a node.* 101
4.41 *Information on a processor object.* 101
4.42 *The* **Thread View**. 103
4.43 *The* **Status Information View**. 104
4.44 *Auxiliary view showing information of a function.* 105
4.45 *The* **Navigation View**. 107
4.46 *The* **RPC Statistics View**, *with the option menu popped up.* 109
4.47 *Auxiliary view on the number of RPCs from one PO to another.* 110
4.48 *The* **Processor/Processor-Object Activity View**, *with its option menu popped up.* 111
4.49 *Auxiliary view from the* **Processor/Processor-Object Activity View**.112
4.50 *Program execution in a processor-object.* 113
4.51 *Two processor-objects executing on a processor.* 114
4.52 *The* **Function Usage View**. 116

4.53 *Selecting a function in the* **Function Usage View** *brings up further information.* 117

4.54 *The* **Composite Function View**. 118

4.55 *The auxiliary view from the* **Composite Function View**. 118

5.1 *The* **Thread View** *of the simple master-slave program.* 126

5.2 *Merge-sort using a 4-level binary tree. The left figure shows the initial placement of the nodes on processors, and the right figure the optimised placement.* 127

5.3 *The* **Processor/Processor-Object Activity View** *reveals the inefficiency of processor usage.* 128

5.4 *The merge-sort program is heavy with RPC activity.* 129

5.5 *Higher efficiency in processor usage after program tuning.* 130

5.6 *The* **Thread View** *shows that the master-PO is idle while the slave-POs are executing.* 132

5.7 *The* **Source-Code View** *shows the source-code area where the master-PO is blocked.* 133

5.8 *The* **Thread View** *of* **Implementation-2**, *with the compute-thread.* . 134

5.9 *The interrupt-signaling mechanism in work.* 136

5.10 *Three-tier architecture for the electronic transaction system.* 138

5.11 *The* **Thread View** *shows the unfairness of transactions.* 139

5.12 *Auxiliary view showing a transaction request being handled by a TS-PO on the machine "achilles".* 141

5.13 *The* **Source-Code View** *reveals the inadequacy of the implementation of the transaction server.* 141

5.14 *The* **Function Usage View** *reveals those functions which are heavily used or take much time to execute.* 143

5.15 *Together with the* **Source-Code View**, *the* **Class Hierarchy View** *shows that the time-stamp subsystem does not depend on the implementation of other classes.* 144

5.16 *The* **Class Information View** *of the class "TimeStampClass", displayed upon clicking the associated node on the* **Class Hierarchy View**. 144

5.17 *The* **Composite Function View** *shows which functions can be optimised for each processor-object.* 145

5.18 *This auxiliary view is the result of selecting the longest function bar in the second row from the top of the* **Composite Function View**. . . . 146

5.19 *The* **Function Usage View** *now reveals that the functions are within reasonable frequency and average time of execution.* 147

5.20 *The* **Function Usage View** *reveals approximately similar function profiles to those in Implementation-3.* 150

B.1 *Definition of the transaction request class.* 168

B.2 *The customer database class.* 168

B.3 *The stock database class.* 169

B.4 *Definition of the business logic unit.* 170

B.5 *Definition of the transaction server.* 171

B.6 *Definition of the transaction resolution subsystem.* 172