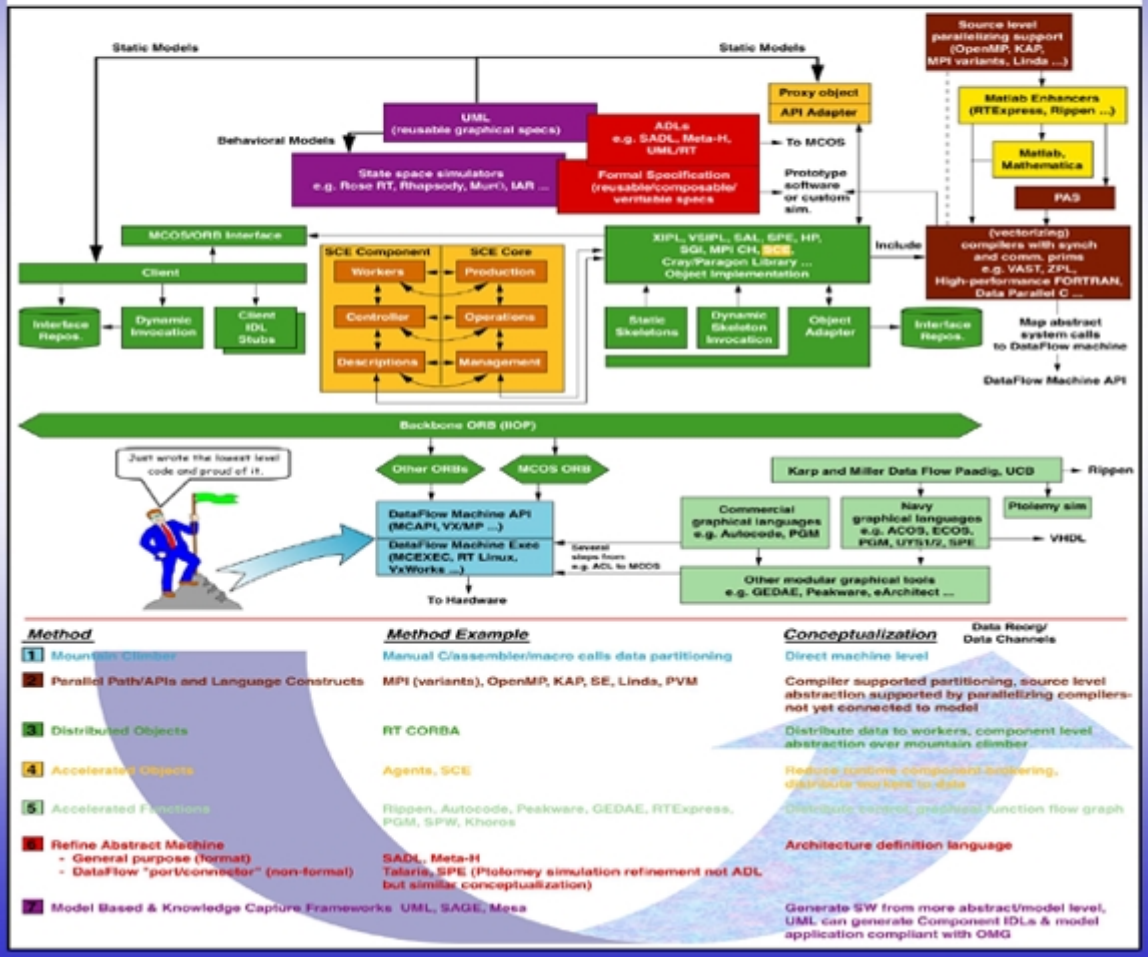
Model Based Parallel Programming with Profile-Guided Application Optimization

Dr. Jeffrey E. Smith
Mercury Computer Systems
jesmith@mc.com

Dr. David Kaeli
Northeastern University
kaeli@ece.neu.edu

Methods to Conceptualize/Apply High Performance Data Flow Applications



Method	Method Example	Conceptualization
1 Mountain Climber	Manual C/assembler/macro calls data partitioning	Direct machine level
2 Parallel Path/APIs and Language Constructs	MPI (variants), OpenMP, KAP, SE, Linda, PVM	Compiler supported partitioning, source level abstraction supported by parallelizing compilers- not yet connected to model
3 Distributed Objects	RT CORBA	Distribute data to workers, component level abstraction over mountain climber
4 Accelerated Objects	Agents, SCE	Reduce runtime component bindering, distribute work/data to data
5 Accelerated Functions	Rippen, Autocode, Peakware, GEDAE, RTExpress, PGM, SPW, Khoros	Distribute control, graphical function flow graph
6 Refine Abstract Machine - General purpose (format) - DataFlow "port/connector" (non-formal)	SADL, Meta-H Talaria, SPE (Ptolomey simulation refinement not ADL but similar conceptualization)	Architecture definition language
7 Model Based & Knowledge Capture Frameworks UML, SAGE, Mesa		Generate SW from more abstract/model level, UML can generate Component IDLs & model application compliant with OMG

Data Reorg/ Data Channels

Problems with Described Development Approaches

- Development and maintenance costs associated with Method 1
- Conceptualizations/tools represent computation (e.g. graph) or communication (e.g. VI) model
- Lack of UML data flow support
- Multiple architecture and library standards to call functions with same signatures
- ADL application in streaming, high performance, data-flow domain
- Perception of inefficiency

An example of a Profiling System: DSPTune for the SHARC DSP Family

- A set of library routines that enable the user to instrument C and assembly programs
- Function calls can be inserted at various locations in the application code, enabling execution driven simulation and instrumentation
- The user provides:
  - •instrumentation routines, which specify the selected instrumentation events (e.g., loads, branches, traps)
  - •analysis routines, which carry out the desired simulation (e.g., caches, stacks, branch predictors)
- Latest version (BDSPTune) allows the user to directly modify the binary ELF files

Cache Line Coloring call graph edges (A-B, B-C, A-D, C-D)

A	A	A	A	D	D	D	D
1	2	3	4	1	2	2	4
		C	C	B	B	B	
		1	2	1	1	2	

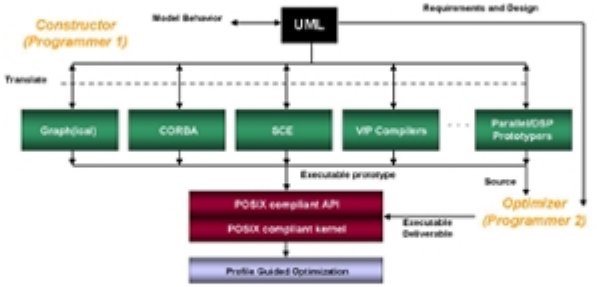
Cache Size

No Conflicts

Observations

- UML doesn't include data flow … yet
- You can translate UML diagrams to any source - might be an avenue of tool support worth exploring
- Specifications (signature) of varied libraries constant
- Graph notation deterministic when combined with ADL target to parallel machine - distributes itself based on queue information
- The trade between block and graph language graphical techniques is that GEDAE-like tools use fixed time line scheduling vs PGM-like tools who stick to the data flow model for run time flexibility
- All of the graphical (light green) techniques shown outgrowth from seminal paper, R.M. Karp and R. E. Miller dating from 1961

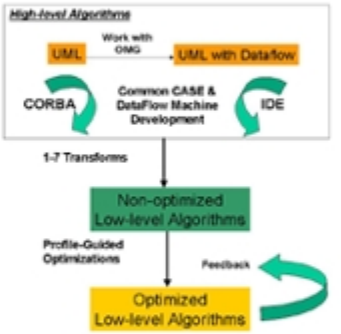
Goals: Component Reuse, Software Productivity, Leverage Existing Investments and Wider Programming Base

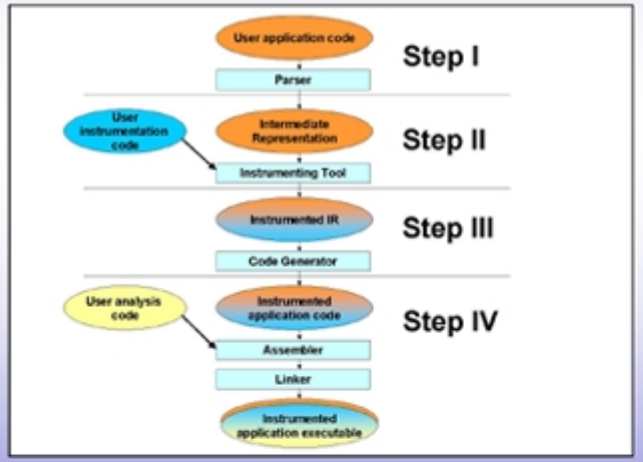


Dynamic Compilation Can Provide a Solution

Collect runtime execution behavior

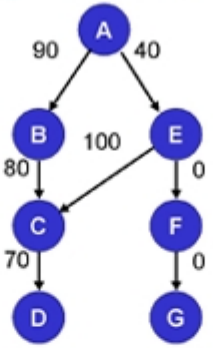
- Memory usage
  - instruction and data caches
  - translation lookaside buffers
- Control flow
  - branch probabilities
  - program "traces"
- Call graphs
  - gprof statistics
- Data dependencies
  - data-dependent control flow
- Variable values
  - value locality
  - interprocedural dataflow
- Hardware counters
  - pipeline stalls





Dynamic Compilation Model is Well-suited for the High Performance Embedded Computing Environment

- Profiles can be used to:
  - Generate control and dataflow graphs
  - Identify program "hot-spots"
  - Reorganize code and data
  - Selectively apply aggressive compilation techniques
    - procedure inlining
    - loop unrolling
    - procedure specialization
    - procedure cloning
  - Reschedule code



An Example of A Dynamic Compilation System Cache Line Coloring

- Attempts to reorder a program executable by coloring the cache space, avoiding caller-callee conflicts in a cache
- Can be driven with both static call graphs and profile data
- Improves upon the work of Pettis and Hansen by considering the organization of the cache space (i.e., cache size, line size, associativity)
- Can be used with different levels of granularity (procedures, basic blocks) and applied both intra- and inter- procedurally
- Programs can be sped up by as much as 100%

Next Steps

- Application to IR formation, fusion, template matching
- Collect software productivity metrics on above and MITRE benchmarks
- Experiment with optimization of UML transformed (through data parallel CORBA or specialized data parallel compiler IDEs) software to efficient Mercury platforms
- Work with OMG in introducing Data Flow, in a way that supports streaming high performance data flow distributed computers (see us for viewgraphs)
- Examine possibility of embedding dynamic profile optimization into run-time system
- Work with CASE and IDE vendor to integrate model-based development of efficient streaming high performance data flow distributed computer targets

Citations

Analysis of Temporal-Based Program Behavior for Improved Caches Performance", J. Kalamatianos, A. Khalafi, D. Kaeli and W. Meleis, IEEE Transactions on Computers, Vol. 10, No. 2, February 1999, pp. 168-175.

"Characterization, Tracing and Optimization of Commercial I/O Workloads", H. Huang, M. Teshome, J. Carreira and D. Kaeli, Proceedings of the 1st Workshop on Computer Architecture Evaluation Using Commercial Workloads, January 1998.

"Efficient Procedure Mapping using Cache Line Coloring", A.H.Hashemi, D. Kaeli and B. Calder, Proceedings of ACM SIGPLAN Conference on Programming Languages Design and Implementation, June 1997, Las Vegas, Nevada, pp. 171-182.

"Analysis of Temporal-based Program Behavior for Improved Cache Performance", J. Kalamatianos, A. Khalafi, D. Kaeli and W. Meleis, Special Issue on Cache Memory, IEEE Transactions on Computers, Vol 48, No 2, February 1999, pp. 168-170.

"A Study of Loop Unrolling for VLIW-based DSP Processors", S. Sair and D. Kaeli, Proceedings of the 1998 Workshop on Signal Processing Systems, October 1998, pp. 519-527.

"Welcome to the Opportunities of Binary Translation", E. Altman, D. Kaeli and Y. Sheffer, IEEE Computer Magazine, special issue on Binary Translation, March 2000, pp. 40-45.

S. Dwivedi, J. Smith and T. Hartrum, "Translating Graphically-Based Object-Oriented Specifications to Formal Specifications", submitted for publication in IEEE Transactions on Software Engineering.

"Data Flow for UML", J. Smith, OMG Proposal for RFP, 9/1999.

Computer Systems, Inc.
MERCURY