An Open64-based Framework for Analyzing Parallel Applications

Laksono Adhianto[1] Barbara Chapman[2]

[1] Department of Computer Science
Rice University

[2] Department of Computer Science
University of Houston

Open64 Workshop, 2008

Outline

1. Introduction
   - Motivation and Objectives
2. Methodology
   - Aproach, implementation and applications
3. Conclusion

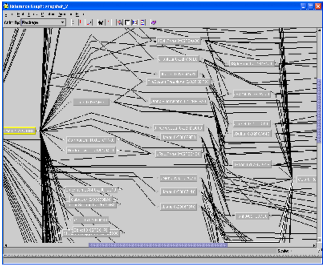
Motivation: Analyzing Complex Applications

Murphy's law:

> *Program complexity grows until it exceeds the capabilities of the programmer who must maintain it.*

Hello World

From a simple sequential program . . .



To a complex large scale application [a]

[a] Image courtesy of javaworld

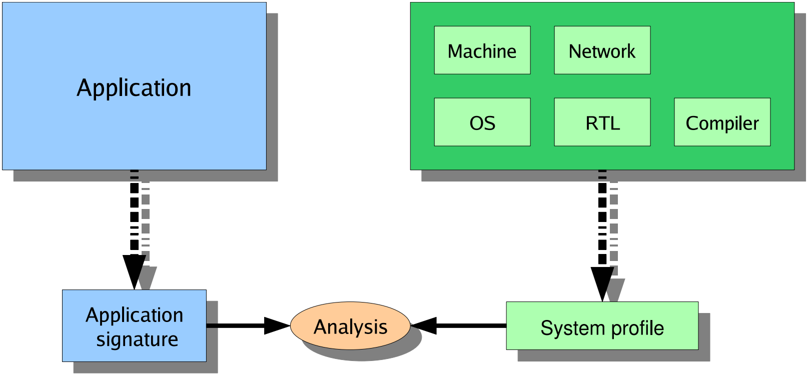
Objectives

We need an infrastructure for

- Understanding large-scale parallel MPI/OpenMP applications.
- Performance modeling and prediction.
- Program optimization and program correctness verification.

Approach

- Extract program skeleton based on **compiler** analysis.
- Retrieve information on communication latency and parallelization overhead from **microbenchmarks**.



Methodology

- Using compiler technology to analyze the source code and microbenchmarks to probe system profile.

$$Analysis = Application\_Signature \otimes System\_Profile$$

- *Application signature*: characterizes the fundamental aspects of an application independent of the machine where it executes
(definition borrowed from PERC SciDAC project).
- *System profile*: characteristics of the platform where the application will be executed.

Application Signature vs. System Profile

System profiles

Application signatures

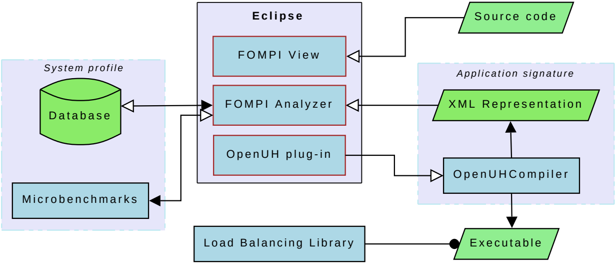
	System 1	*System 2*	*System 3*	...
Application 1	Ω_{11}	Ω_{21}	Ω_{31}	...
Application 2	Ω_{12}	Ω_{22}	Ω_{32}	...
Application 3	Ω_{13}	Ω_{23}	Ω_{33}	...
...	...	...	...	...

- *Application signature*: independent to system configuration
- *System profile*: independent to application programs

FOMPI Framework

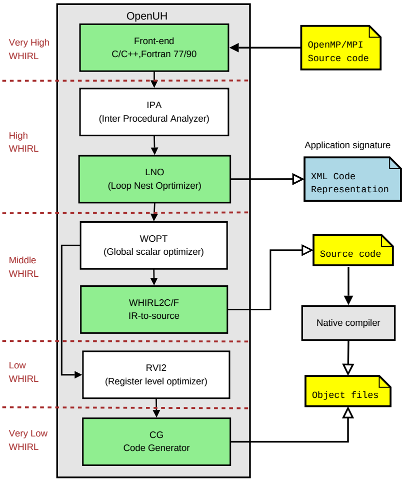
FOMPI: **F**ramework for analyzing **O**penMP and **MPI** applications.

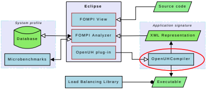
$$Analysis = Application\_Signature \otimes System\_Profile$$



References:
Parallelization [1, 5], OpenMP tools [7, 6], Compiler [8], Modeling [4, 3], Autoscoping [2].

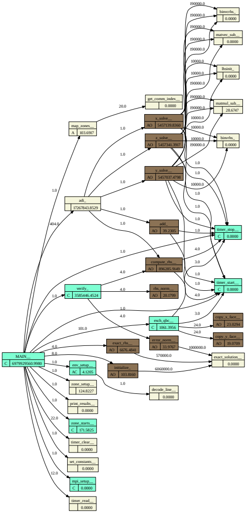
The OpenUH Compiler





- Rich of analysis: data dependence, inter-procedural analysis, array region analysis, . . .
- We have extended OpenUH for generating *application signature*.
    - Containing MPI routines, OpenMP, loops, estimated execution time, cache access pattern, . . .

Call Graph



Traditional call graph is not scalable for large scale applications containing million lines of code.

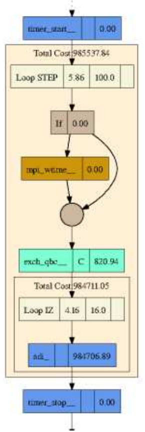
$$G_c = \langle N_c, E_c, s_c \rangle$$

- Aggregated execution time:

$$N_c^i = N_c^x + \sum_{i=0}^{n} \left(N_{c_i}^i \times E_c^f \right)$$

Inclusive execution time of a unit is the sum of exclusive time and the total time of its call sites.

Control-Flow Graph



$G_f = \langle N_f, E_f, s_f \rangle$

Inclusive cost:

$$N_f^i \quad = \quad \begin{cases} N_{f_l}^x + \sum_{i=0}^{n}(E_{f_i} \times N_{f_i}^i), & \text{inside loop;} \\ N_{f_b}^x + \max_{i=0}^{n}(\sum_{j=0}^{m_i} N_{f_{i_j}}^i), & \text{branches;} \\ N_f^x, & \text{otherwise.} \end{cases}$$

Exclusive cost:

$$N_{f_l}^x = \begin{cases} t_{ser}^{comp} = E_f \times (t_{machine} + t_{overhead} + t_{cache}), & \text{serial} \\ t_{par}^{comp} = \frac{t_{ser}^{comp}}{n_t} + \sum t_{unpar}^{comp} + \sum O, & \text{parall} \end{cases}$$

Summary

- FOMPI provides portable and scalable analysis and modeling with no program execution needed
  - Based on *application signature* from the compiler and *system profile* from microbenchmarks.
- Applications include: performance modeling, program understanding, OpenMP generation and MPI load imbalance reduction.
- Open64's extensibility is needed: WHIRL, analysis and transformation
  - SUIF, GCC GEM Framework, LLVM . . .

Contributions

Our contributions:

- Compiler extension to extract *application signature*
- Microbenchmark extensions for more OpenMP coverage
- Using compiler and microbenchmarks for program analyses and modeling
- Scalable call graph and control-flow graph
- Runtime library for reducing load imbalance

Acknowledgments

- John Mellor-Crummey (Rice University)
- HPCTools members and alumni
- TLC^2
- PSTL lab
- UH, CS@UH
- PModels, DOE and NSF

For Further Reading I

L. Adhianto, F. Bodin, B. Chapman, L. Hascoet, A. Kneer, D. Lancaster, I. Wolton, and M. Wirtz.
Tools for OpenMP application development: the POST project.
Concurrency: Practice and Experience, 12(12):1177–1191, 2000.

Laksono Adhianto and Barbara Chapman.
Autoscoping support for openmp compiler.
In *Workshop on Tools and Compilers for Hardware Acceleration*, 2006.

For Further Reading II

- Laksono Adhianto and Barbara Chapman.
  Performance modeling of communication and computation in hybrid mpi and openmp applications.
  International Conferences on Parallel and Distributed Systems (ICPADS)-Workshop of Performance Modeling and Analysis of Communication (PMAC), 2:3–8, 2006.
- Laksono Adhianto and Barbara Chapman.
  Performance modeling of communication and computation in hybrid mpi and openmp applications.
  Simulation Modelling Practice and Theory, 15(4):481–491, 2007.

For Further Reading III

- Laksono Adhianto and Michael Leuschel.
  Strategy for improving memory locality reuse and exploiting hidden parallelism.
  In *Indonesian Students Scientific Meeting (ISSM)*, Manchester, UK, August 2001.

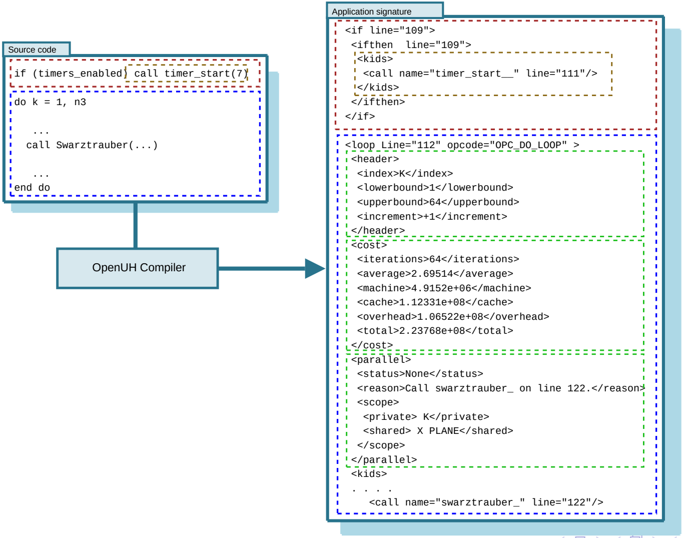
- Barbara Chapman, Oscar Hernandez, Lei Huang, Tien-hsiung Weng, Zhenying Liu, Laksono Adhianto, and Yi Wen.
  Dragon: An open64-based interactive program analysis tool for large applications.
  In *4th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, 2003.

For Further Reading IV

- Barbara Chapman, Tien-Hsiung Weng, Oscar Hernandez, Zhenying Lui, Lei Huang, Yi Wen, and Laksono Adhianto.
  Cougar: Interactive tool for cluster computing.
  In *Proceedings of the 6th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI'2002)*. The International Institute of Informatics and Systemics, 2002.

- OpenUH.
  http://www.cs.uh.edu/õpenuh.

Application Signature: Example from NAS FT



Source code

```
if (timers_enabled) call timer_start(7)
do k = 1, n3

   ...
  call Swarztrauber(...)

   ...
end do
```

Application signature

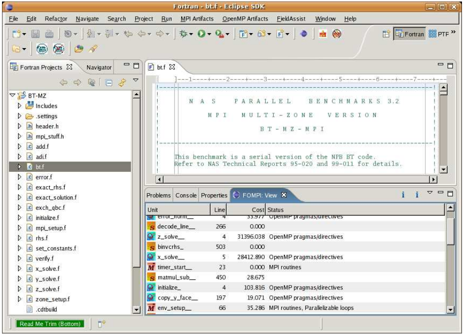
```
<if line="109">
 <ifthen  line="109">
  <kids>
   <call name="timer_start__" line="111"/>
  </kids>
 </ifthen>
</if>
<loop Line="112" opcode="OPC_DO_LOOP" >
 <header>
  <index>K</index>
  <lowerbound>1</lowerbound>
  <upperbound>64</upperbound>
  <increment>+1</increment>
 </header>
 <cost>
  <iterations>64</iterations>
  <average>2.69514</average>
  <machine>4.9152e+06</machine>
  <cache>1.12331e+08</cache>
  <overhead>1.06522e+08</overhead>
  <total>2.23768e+08</total>
 </cost>
 <parallel>
  <status>None</status>
  <reason>Call swarztrauber_ on line 122.</reason>
  <scope>
   <private> K</private>
   <shared> X PLANE</shared>
  </scope>
 </parallel>
 <kids>
 . . . .
    <call name="swarztrauber_" line="122"/>
```

Application Signature: Scalability

- Stored in XML file: XML Program Representation.
- Designed for interoperability and scalability in mind.

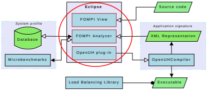
	LOC	loops	XML	Tags	Tags/Loop	XML/LOC
ammp	10068	272	11872	6767	24.88	1.18
applu	2555	120	3497	2585	21.54	1.37
apsi	4386	278	9582	6815	24.51	2.18
art	1570	72	2672	1799	24.99	1.70
equake	1121	70	2338	1650	23.57	2.09
gafort	720	72	2450	1748	24.28	3.40
mgrid	1023	48	1338	1009	21.02	1.31
swim	275	24	677	505	21.04	2.46
wupwise	1018	43	2096	1419	33.00	2.06
Average	2526.22	111.00	4508.00	3257.00	24.31	1.97

Eclipse



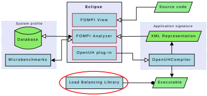
```
!-------------------------------------------------------------------------!
!                                                                         !
!        N A S     P A R A L L E L     B E N C H M A R K S  3.2          !
!                                                                         !
!            M P I    M U L T I - Z O N E    V E R S I O N                !
!                                                                         !
!                          B T - M Z - M P I                              !
!                                                                         !
!-------------------------------------------------------------------------!
!                                                                         !
!    This benchmark is a serial version of the NPB BT code.               !
!    Refer to NAS Technical Reports 95-020 and 99-011 for details.        !
!                                                                         !
```

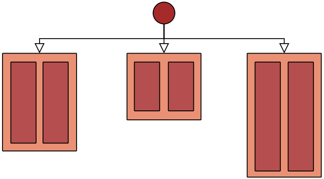
Unit	Line	Cost	Status
error_norm__	4	33.977	OpenMP pragmas/directives
decode_line__	266	0.000	
z_solve__	4	31396.038	OpenMP pragmas/directives
binvcrhs_	503	0.000	
x_solve__	5	28412.890	OpenMP pragmas/directives
timer_start__	23	0.000	MPI routines
matmul_sub__	450	28.675	
initialize_	4	103.816	OpenMP pragmas/directives
copy_y_face__	197	19.071	OpenMP pragmas/directives
env_setup__	66	35.286	MPI routines, Parallelizable loops



- Eclipse is an open source platform based on IBM VisualAge Micro Edition.
- We have developed FOMPI view plugin as the main user interface
    - Accessible by any Eclipse *perspective*.
    - Can generate call graph, control-flow graph, . . .

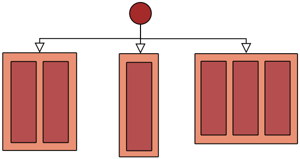
MPI Load Imbalance





Unbalanced program

- Each MPI process has the same number of OpenMP threads.



Balanced program

- Adjust the number of OpenMP threads according to the workload.

MPI Load Imbalance

- Assumption: application includes a main iterative loop containing:
  - Large computation
  - Significant communication
- Approach:
  1. Statically determine the main iterative loop
  2. Insert load balancing library (LBL) at the beginning and at the end of the loop.

MPI Load Imbalance

Original code

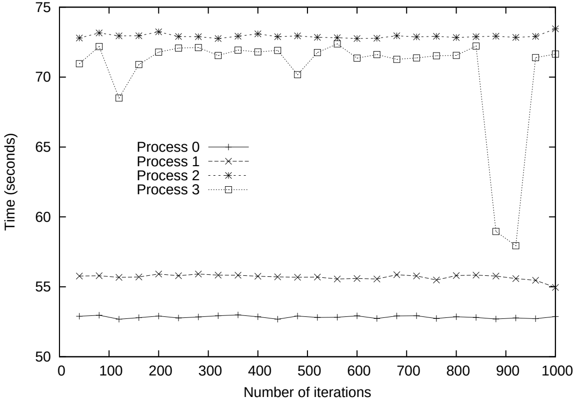
```
#include <lbl.h>
int MainFunction () {
          . . .
  /* Main iteration */
  while (iter<MAX_ITER) {
    . . .
    Do_Computation ();
    . . .
    Do_Communication ( ) ;
  }
 /* end of main iteration */
          . . .
}
```

With Load balancing library

```
#include <lbl.h>
int MainFunction () {
  LBL_SetSkipIterations(40);
  LBL_SetThreshold(30);
  LBL_Init();
  /* Main iteration */
  while (iter<MAX_ITER) {
    LBL_LoadDetection ();
    . . .
    Do_Computation ();
    . . .
    Do_Communication ( ) ;
 }
 /* end of main iteration */
 LBL_Finalize ( ) ;
}
```

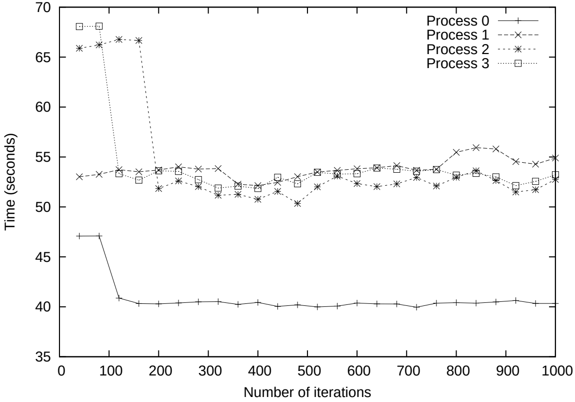
MPI Load Imbalance

Original code with load imbalance



MPI Load Imbalance

Using load imbalance library


Process 0
Process 1
Process 2
Process 3
Time (seconds)
Number of iterations