PACE
The Warwick Performance System

G.R. Nudd, E. Papaefstathiou, D.J. Kerbyson,
High Performance Systems Group
University of Warwick
- Background
- Major European Collaboration Project
- Application Driven
- PERFORMANCE ANALYSIS & CHARACTERISATION ENVIRONMENT
Aim

Develop techniques to enable users.application developers to make efficient use of distributed resources

- Application Development
- System Resources
- On the fly Optimization
Applications

- Graphics (image synthesis)
- Image Processing
- Real Time Signal Processing
- Financial Codes
On-Going Research

- System Overview
  - Methodology
  - Performance Language
  - Application Workloads

- Models & Outputs
  - Model Development
  - System Outputs
  - On-the-fly & Scheduling
Performance Oriented Software Development (not!)
Software Performance Studies

- Design/Algorithm
- Sequential Code
- Parallel Code

Prediction Studies

Capacity Planning & Tuning

Target Platform

Characterization & Traditional Modeling

Monitoring
Goals
Support Software Lifecycle

Requirements
- Performance bounds, scalability

Design
- Parallelisation selection

Implementation
- Performance understanding

Execution
- Dynamic optimization
- Capacity planning, static optimization

Maintenance

Software Model
Goals: Open System

Evaluation

Workload

3rd Party H/W Models

Custom H/W Models

Custom analysis tools

3rd party analysis tools
Goals

- Work with any system
- Multiple level of abstraction
  - Accuracy
  - Speed of evaluation
  - Model development effort
- Automation
Layered Framework

Description

Prediction System

Performance Analysis

Application Layer

Parallel Template

Hardware Layer

2 4 8 16
CPU  Memory  Network  I/O

100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%

2 4 8 16
CPU  Memory  Network  I/O

Description Prediction System Performance Analysis

Layered Framework

Application Layer

Parallel Template

Hardware Layer

2 4 8 16
CPU  Memory  Network  I/O

100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%

2 4 8 16
CPU  Memory  Network  I/O
Hierarchical Layered Framework Diagrams (HLFD)

Application Layer

Subtask-1 Model
Subtask-2 Model
Subtask-3 Model
Subtask-4 Model

Parallel Template Layer

Parallel Template-1
Parallel Template-2
Parallel Template-3

Hardware Layer

Hardware Model-1
Hardware Model-2
Application Layer

Hv

Resource Usage Vector
flops, language
operations, instructions, memory
references, etc

Hardware Model
statistical, analytical, simulation

\[ T_x = hm(v_{c1}) + L1 \cdot (hm(v_{c2}) + P1 \cdot hm(v_{c3}) + P2 \cdot hm(v_{c4})) \]
Parallel Template
Overview of PACE

Workload
- Source Code Analysis
- Object Editor
- Object Library

Core System
- Language Scripts
- Compiler

H/W Models
- CPU
- Cache
- Network
- Hardware Models
- Evaluation Engine
- Application Model

Performance Analysis

Outputs
- On the Fly
PACE

- Is based on a performance language
  - Control flow, parallelisation, resource usage
- Automated model creation
- Automated model evaluation
- Performance analysis environment
- Other support tools (e.g. source code analysis, object browser, visualization)
Core System

Workload
- Source Code Analysis
- Object Editor
- Object Library

Compiler

Language Scripts

H/W Models
- CPU
- Network
- Cache
- Evaluation Engine
- Application Model

Performance Analysis
On the Fly

Outputs

Core System
Model Objects

Model Object Structure

<table>
<thead>
<tr>
<th>application</th>
<th>user input</th>
<th>system &amp; problem setup</th>
<th>subtasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtask</td>
<td>problem size</td>
<td>computation (cflow)</td>
<td>partmp</td>
</tr>
<tr>
<td>partmp</td>
<td>problem size &amp; computations</td>
<td>model config</td>
<td>hardware models</td>
</tr>
</tbody>
</table>

```
application improc {
...
link {
    histo:
    SizeX = ImgSzX/hardware.Nproc,
    SizeY = ImgSzY;
...
}
proc exec init {
    ...
    call histo;
    ...
}
```
Control Flow Procedures

proc cflow Merge{
  var vector:
  start = <is ct, 1.47>,
  merge = <is suif, ldc_i32, add_i32>,
  tlcp = <is ct, 6.03>,
  otcp = <is clc, IASG, ICMP,
          2*IADD, 4*VIDX, 2*FASG>;
  compute start; (* C1 *)
  loop (<is clc, FOR>, 2*dtpp*Pmrg) (* L1 *)
    compute merge; (* C2 *)
  loop (<is clc, FOR>, 2*dtpp*(1-Pmrg)) (* L2 *)
    compute tlcp; (* C3 *)
  loop (<is clc, FOR>, dtpp) (* L3 *)
    compute otcp; (* C4 *)
}

C1
  L1
  C2
    L2
      C3
        L3
          C4
Step and Confdev

1. `step <model> on <procs> {`
   - Insert event on processor event list
   - Initialise underlying model

2. `confdev <parameters>`
   - Configure model

3. `}`
   - Evaluate model (in some cases)

```
step cpu on 2,hardware.Nproc {
    confdev TxProc;
}

step pvminitsend on 2,hardware.Nproc {
    confdev PVM_DataDefault;
}

step pvmpack on 2,hardware.Nproc {
    confdev MsgSize,PVM_Int;
}

for( i = 2; i <= hardware.Nproc; i++ )
    step pvmcom {
        confdev i,1;
    }
```
Source Code Analysis

- Use SUIF to extract computational and memory resources
- Profiler, and manual specification, of looping & branching
When considering code characterisation, SUIF offers different levels of detail:

- **High Level Code Characterisation**
  - Automation, architecture neutral
  - Inaccurate, language specific

- **Intermediate Code Characterisation**
  - Language and architecture independent
  - Accurate, easy to automate, optimisation
  - Support for C, Fortran, C++, and Java

- **Instruction Code Characterisation**
  - Accurate, language neutral
  - Difficult to automate, system dependent
Manual Workload Definition

- **When?**
  - Source code not available (e.g. design stage)
  - Computational code extracted from sequential code
  - Experimentation in a high level of abstraction

- **Tools**
  - Workload Definition Environment
  - Libraries of objects
Workload Definition Environment
Hardware Models

Source Code Analysis
Object Editor
Object Library

Language Scripts
Compiler

H/W Models
CPU
Cache
Network

Evaluation Engine
Application Model
Hardware Models

Performance Analysis
On the Fly

Outputs
Hardware Models

- Individual models for system components
  - Modularity
  - Multiple levels of abstraction
- Workload interface
- Composite models in hardware layer
Component Models

- Each component model contains:
  - interface to workload (input)
  - configuration parameters
  - evaluation method (output time prediction)

- Encapsulate detail
- Plug In
Type of Models

- Focused on generic models
  - common characteristics and heuristics
- Analytical based
- Models include:
  - CPU
    » C, Fortran (Language construct costs)
    » SUIF (Intermediate format costing)
  - Network
    » MPI, PVM (message passing interfaces)
  - Cache
    » Direct mapped, Set associative, L1, L2
Example: Cache Model

- Hybrid
- Multi-level
  - primary, secondary
- nested loop structures
- Fast
  - orders quicker than sim.
- Accurate
  - <10% error on Miss ratio
Cache Model - approach

- Multi-level
- Each level acts as a filter
- Workload refined between levels
Cache - Workload

- Workload (extracted from code)

```
DO I = 0, N-1
  DO J = 0, N-1
    Z(J, I) = 0.0
  DO K = 0, N-1
    Z(J, I) += X(K, I) * Y(J, K)
  ENDDO
ENDDO
ENDDO
```

- Refined using cache ‘Footprints’

```
DO J = 0, M-1
  DO I = 0, N-1
    X(I) = Y(I)
  ENDDO
```

array X:double[N,N] at 0
array Y:double[N,N] next
array Z:double[N,N] next
loop 0, N-1
loop 0, N-1
  Z[loop(2), loop(1)]:w
  loop 0, N-1
  X[loop(3), loop(1)]
  Y[loop(2), loop(3)]
  Z[loop(2), loop(1)]:rw
Hardware Model Configuration Language (HMCL)

- HMCL specifies target system
- Combination of:
  - component performance models
  - configuration parameters
  - connectivity
- Heterogeneous
- Possible to consider dynamic changes
Example HMCL description

component computer Sun4 {
  cpu_clc {
    IASG = 2.2,
    DADD = 4.1,
    ...
  }
  cache_l1 {
    Capacity = 16384,
    Assoc = 2, LSize = 64;
  }
  ...
}

component network Ethernet {
  PVM {
    TxA Send(l) = 539 + 23.l,
    ...
  }
  ...
}

Sun4 ClusterA[32];
Ethernet ANet;

connect ClusterA[1:32] to ANet;

- Two components:
  Sun4,
  Ethernet

- performance models:
  cpu_clc,
  cache_l1,
  PVM

- 32 nodes on Ethernet
PACE - Outputs

**Workload**
- Source Code Analysis
- Object Editor
- Object Library

**Language Scripts**

**Compiler**

**H/W Models**
- CPU
- Cache
- Network

**Performance Analysis Outputs**

**On the Fly**

**Hardware Models**
- Evaluation Engine
- Application Model

**Core System**
Performance Model Decomposition

- Parameters include:
  - Application
  - System specification

- Input parameters visible to objects (derive internal variables)

- Parameter specification creates scenarios
Scalability Analysis

Example: Financial Option on Network Cluster
Time Decomposition

- Hierarchical

- Decomposition into:
  - System
    - CPU, cache, network, I/O
  - Application
    - sub-tasks, procedures, etc.

- Group Components
  - identify bottlenecks
Predictive Traces

- Generate events from workload information
- Integration with measurement tools for visualization
  - Paragraph (PICL format, ORNL)
  - Pablo (SDDF format, UIUC)
Example visualization
Predictions On-the-fly

- Fast prediction evaluation
- Pre-execution evaluation
  - Algorithmic choices
  - System configuration
- Extend to:
  - Multi-task Scheduling
  - Dynamic task partitioning (& load balancing)
  - Performance instrumentation
- Principal:
  - Add performance model to application executable
Performance ‘stub’

User Parameters
{ Problem, System }

Implementation Choice

Code 1

Code 2

User directed execution

System

= User decision

Performance directed execution

User Parameters
{ Problem }

Performance Model

Code 1

Code 2

Single Executable

System
Example (Algorithmic choice)

- Image processing convolution

- Problem parameters - image size, filter size

- Parameters determined by performance model:
  - processing method
  - #Processors (System - SUN Ultra Cluster)
Model output

Max: 8 workstations  Max: 16 Workstations

(Numbers in each box indicate #workstations to use for given problem parameters)

http://www.dcs.warwick.ac.uk/~hpsg/reference_top.htm
Schedule Optimization

- Use performance model associated with each task
- Pre-execution evaluation
  - time prediction
  - best system mapping
- Optimize schedule
Mutiprocessor Scheduling

Tasks
\{T_0, T_1, ..., T_{n-1}\}

Processors
\{P_0, P_1, ..., P_{m-1}\}

For each task:
- \(\beta \in \{0, ..., m - 1\}\) Set of processors
- \(t_j \quad \tau_j\) Start time, and duration

Goal: minimise the makespan
\[
\max_{0 \leq j \leq n-1} \{t_j + \tau_j (|| \beta_j ||)\}
\]

- Time prediction not available in traditional scheduling
Schedule makespan Improvement

- Genetic Algorithm Approach -
  - Evolutionary method
    - One-time start-up cost
    - Small evolution cost (additional tasks/ changing resources)

Example - 15 tasks 10 processors

![Graph showing makespan vs schedules tried for random search and genetic algorithm methods.]
Applications

- Financial Options
  - Monte-Carlo Simulation, PDE (UK Banks)

- Image Processing
  - Real-time image processing (NPL, London),
  - SAR (NA-Soft),

- Embedded Signal Processing
  - Signal Processing (Thomson ASM, Nice)

- Graphics
  - Photo-realistic generation (Mental Images, Berlin)
On-going research

- Shared memory
- Heterogeneous
- Object Orientated
- on-the-fly

http://www.dcs.warwick.ac.uk/~hpsg