A Mixture of Experts Approach for Runtime Mapping in Dynamic Environments

Murali Emani
School of Informatics
University of Edinburgh
Modern computing hardware

Diverse
Stochastic
Evolving
Parallelism Mapping

Program

Computation Steps

Hardware
Parallelism Mapping

- Workloads
- Software
- Data
- Hardware

Program

Program performance is sensitive to the environment
What exactly is the problem?

Optimal partitioning of the parallel work is

*not static* and *non-trivial*
What exactly is the problem?

Existing approaches are based on one-size-fits-all policy.
What exactly is the problem?

Existing approaches are based on **one-size-fits-all** policy

- Not suitable for dynamic environments
- Hard to extend and update
Goals

➔ Determine optimal resources for a parallel program

\textit{Avoid under-subscription / over-subscription}

➔ Enable program auto-tuning

\textit{Adapt smartly to varying resources}

➔ Program and Platform aware

\textit{Generic and portable}
Where does it fit in the stack

Application

Runtime

Operating System

Hardware
State Space
Idea

- Identify best mapping policy in each set
Idea

- Identify best mapping policy in each set

\[ E^k \rightarrow E^{k-1} \rightarrow E^1 \rightarrow E^2 \]
Idea

Collect these policies

$E^k$ $E^1$

$E^{k-1}$ $E^2$

$E^1$

$E^2$

$E^{k-1}$

$E^k$
Idea

- Choose the best policy based on current state
Idea

→ Choose the best policy based on current state
Mixture of Experts based Mapping

- Ensemble of experts (mapping policies)

- Smart way to select the best expert at runtime

- Combine offline prior models with online learning
Mixture of Experts based Mapping

Expert 1

# threads

Expert 2

# threads

...

...

Expert k

# threads
Mixture of Experts based Mapping

How to select the best expert?

Expensive to evaluate with # threads of all experts
Mixture of Experts based Mapping

How to select the best expert?

Expensive to evaluate with # threads of all experts

Environment predictor
Mixture of Experts based Mapping

How to select the best expert?

Expensive to evaluate with # threads of all experts

Environment predictor
Predictive Modelling

Thread predictor

\[ w(f_t) = \hat{n}_t \]

What is the best number of threads

Environment predictor

\[ m(f_t) = e_{t+1} \]

What should the environment look like
Predictive Modelling

Thread predictor

\[ w(f_t) = \hat{n}_t \]

What is the best # threads

Environment predictor

\[ m(f_t) = e_{t+1} \]

What should the environment should look like

Input-feature-vector = \(<\text{code, environment}>\)

\[ f = (c,e) \]
Approach – Machine Learning
Hand crafted solutions infeasible

Approach – Machine Learning

Training data → Data Pre-processing → Learning algorithm → Model → prediction

New input
Approach – Machine Learning

- Hand crafted solutions infeasible

- Train offline, deploy online
- Supervised learning, Cross-validated
- Trained on NAS, evaluated on additional benchmarks

* Training overhead: one-off cost of 9216 experiments
Training phase

- Various configurations of program pairs and # threads
  9216 experiments; 3 weeks for runs; 1.1 GB log

- Feature space dimensionality reduction: Information gain
  10 / 154 rich subset of features

- Linear Regression Models
## Features

<table>
<thead>
<tr>
<th>STATIC (code)</th>
<th>DYNAMIC (environment)</th>
</tr>
</thead>
<tbody>
<tr>
<td># instructions</td>
<td># workload threads</td>
</tr>
<tr>
<td># branches</td>
<td># processors</td>
</tr>
<tr>
<td># load/store</td>
<td>run queue size</td>
</tr>
<tr>
<td></td>
<td>CPU load</td>
</tr>
<tr>
<td></td>
<td>page free list rate</td>
</tr>
<tr>
<td></td>
<td>cached memory</td>
</tr>
</tbody>
</table>
How to select the best expert

Online Expert Selector

Select expert \( k \)

\[ k : \min \| \hat{e}_k - e_{\text{actual}} \| \]
How to select the best expert

Online Expert Selector

Select expert $'k'$

$$k : \min \| \hat{e}_k - e_{\text{actual}} \|$$

Use 'Environment predictor' as a proxy to select the best mapping policy
All put together...

Online Expert Selector $M$

Input $f = [c, e]$

Experts 1 to $k$

Output $n_{best}$
How many experts?
How many experts?

open question
Started with 4 experts

12 cores (two 6-core Intel E5645)

32 cores (four 8-core Intel Xeon L7555)

E1  E2

E3  E4

scale  do not scale

scaling behavior

Experts
Evaluation

**Platform** : 32-core Intel Xeon

**Benchmarks** : NAS, SpecOMP, Parsec (*OpenMP*)

**Comparison** : OpenMP default, Online, Offline, Analytic

**Workloads** : Small (*light*), large (*heavy*)

**Hardware** : Low, high frequent

---


Results

1.17x over analytic
1.26x over offline
1.38x over online
Why multiple experts? Why not a single model?
Why multiple experts? Why not a single model?

Multiple experts outperforms single model.
Can this approach be used with other optimization techniques?
Can this approach be used with other optimization techniques?

Affinity-based scheduling
To sum up...

Developed an approach for smart parallelism mapping

- Adaptive to dynamic environment
- Predictive modelling at its heart
- Environment predictor as a proxy to select the best mapping policy
What next?

- Integrating this concept in CnC
- Focus on **tuning** component
- Runtime and Application tuning
- Dynamic partitioning of resources to steps
Idea

Instances of computations (steps)

➔ Varying resource requirements for steps
➔ Mapping depends on when data is ready
Take away

⇒ One-size-fits-\textit{none}

⇒ A bag of multiple policies is more practical than one

⇒ Machine learning can be of help !!

Thank you

\textbf{Murali Emani}

University of Edinburgh

m.k.emani@sms.ed.ac.uk
Backup
Adaptive Parallelism Mapping

- Program performance is sensitive to the environment

- Various characteristics
  - Compute/memory/disk bound

- Recurring upgrades
  - Versions compatibility

- Large number of components
  - Increased chances of failure

- Varying amount of I/O
  - Scalability issues
All experts use the same features, they vary in importance across each expert.
The bar chart shows the speedup over default for different scenarios:

- Monolithic: 1.27
- 4 experts: 1.55
- 8 experts: 1.63
Evaluation

**Platform**: 32-core Intel Xeon 4 one-socket nodes, 8 cores/socket, 3.7.10 kernel

**Compiler**: gcc 4.6 “-O3 -fopenmp”

**Benchmarks**: NAS, SpecOMP, Parsec (*OpenMP*)

**Comparison**: OpenMP default, Online, Offline, Analytic

**Workloads**: Small (*light*), large (*heavy*)

**Hardware**: Low, high frequent


What is the effect of increasing # experts?

![Bar chart showing speedup over default for increasing number of experts]

Graceful addition of experts

What about # experts > 4? Needs more analysis