A First Step to Performance Prediction for Heterogeneous Processing on Manycores

ICCS 2015, ALCHEMY Workshop

Nicolas Benoit, Stephane Louise

stephane.louise@cea.fr

CEA LIST/DACLE

June 1, 2015
1 Heterogeneity and the future of computing

2 Selected approach: choice of an execution model

3 Experimental Results

4 Related Works

5 Conclusion and Future Work
Current trends in computing: more core, more heterogeneity.
Current trends in computing: more core, more heterogeneity. Heterogeneous computing provides outstanding advantages

- Less transistors and silicone surface utilization for a given task
Current trends in computing: more core, more heterogeneity.

Heterogeneous computing provides outstanding advantages

- Less transistors and silicone surface utilization for a given task
- More power efficiency
Current trends in computing: more core, more heterogeneity.

Heterogeneous computing provides outstanding advantages

- Less transistors and silicone surface utilization for a given task
- More power efficiency
- Allows for more computing power on a given chip
Toward a Heterogeneous Future of computing

Current trends in computing: more core, more heterogeneity. Heterogeneous computing provides outstanding advantages

- Less transistors and silicone surface utilization for a given task
- More power efficiency
- Allows for more computing power on a given chip

But there are huge drawbacks:
Current trends in computing: more core, more heterogeneity. Heterogeneous computing provides outstanding advantages
- Less transistors and silicone surface utilization for a given task
- More power efficiency
- Allows for more computing power on a given chip

But there are huge drawbacks:
- Hard to program for
Current trends in computing: more core, more heterogeneity.

Heterogeneous computing provides outstanding advantages

- Less transistors and silicone surface utilization for a given task
- More power efficiency
- Allows for more computing power on a given chip

But there are huge drawbacks:

- Hard to program for
- Any application must be retargeted to any new chip
Current trends in computing: more core, more heterogeneity.
Heterogeneous computing provides outstanding advantages

- Less transistors and silicone surface utilization for a given task
- More power efficiency
- Allows for more computing power on a given chip

But there are huge drawbacks:

- Hard to program for
- Any application must be retargeted to any new chip

Ideally speaking a programmer should not be required to know the underlying hardware, and the compiler should do most of the hard work to adapt a given application to a specific target.
There are important milestones and research opportunities toward reaching such a goal:

- Characterizing source code and characterizing fitness of a piece of application to run on a specific core:
  - Characteristics of code (utilization of FP, vector computing, type of control flow, etc.)
  - Data-parallel code or mostly sequential
There are important milestones and research opportunities toward reaching such a goal:

- Characterizing source code and characterizing fitness of a piece of application to run on a specific core:
  - Characteristics of code (utilization of FP, vector computing, type of control flow, etc.)
  - Data-parallel code or mostly sequential

- The compiler must map efficiently the identified sub-parts of an application on a pool of heterogeneous cores.

This is an iterative process with a heuristic to provide a nearly best possible configuration of parallel execution mapping on heterogeneous cores.
In the past we made some research in compilation techniques by defining a IR called Kimble:

- A hierarchical DAG was used as IR capturing both Data Flow, Control Flow, Program hierarchy and computing properties of sub-parts of a program
- The Kimble IR: produced by the Gomet plugin (plugged in the GCC toolchain in the middle-end compiler)
- Allows for code characterization (Amount of processing power, use of floats, vectors, etc.)
- Use pattern matching to associate a part of a program with a possible core for execution
Determining the best configuration for an efficient execution

For a compiler to map efficiently the identified sub-parts of an application on a pool of heterogeneous cores:

- Characterizing the execution time of each sub-part (with possible task coarsening)
- Prediction of the global application for a chosen mapping of the processing parts
- Selection of the best (more efficient) mapping
- An advanced compiler could test several task coarsening configurations to incrementally find the best configuration of the compiled program on the target.
Let us suppose 2 pools of heterogeneous cores: \( n \) cores General Purpose GP, \( m \) cores Specialized Purpose SP

for each partition \( P \) of the total workset

\[
\begin{align*}
\text{prologue (} P \text{); /* GP prologue */} \\
/* request SP processing of a subset of } P /* \text{issue_spec_section (} \text{sp\_subset\_of(} P \text{)});} \\
/* process a subset of } P \text{ on the GPP */ \text{work (} \text{gp\_subset\_of(} P \text{)}); /* GP work */} \\
/* synchronize with SP section execution */ \text{wait\_for\_end\_of\_specialized\_section();} \\
\text{epilogue (} P \text{); /* GP epilogue */}
\end{align*}
\]
Let us suppose 2 pools of heterogeneous cores: $n$ cores General Purpose GP, $m$ cores Specialized Purpose SP

We built a theoretical model of performance based on this Petri net and the parallelism ratio in the sense of Amdahl's law.

A First Step to Performance Prediction for Heterogeneous Processing on Manycores | DACLE Division | June 1, 2015 | 8
Let us suppose 2 pools of heterogeneous cores: \( n \) cores General Purpose GP, \( m \) cores Specialized Purpose SP.

We built a theoretical model of performance based on this Petri net and the parallelism ratio in the sense of Amdahl's law.
A 48 cores’ AMD server system using a mix of The Filterbank and FFT
Experimental results: A SMP standard system with heterogeneous workload

A 48 cores’ AMD server system using a mix of The Filterbank and FFT

Setting the number of GPPs to 12 and the number of SPs to 24
Experimental results on a 48 core SMP

The simulator correctly captures the effect of using a larger number of work partitions: the performance grows as the balancing between GPPs and SPs improves. Also, the prediction mean absolute error is kept very low at 1.59%.
The simulator correctly captures the effect of using a larger number of work partitions: the performance grows as the balancing between GPPs and SPs improves. Also, the prediction mean absolute error is kept very low at 1.59%.
Evaluation on a UltraSparc T1, 1/2

A 8 core system, with 4 threads per core.

Gomet runtime assigns thread/core affinity so that the number of working threads per physical core remains as low as possible.

Workload: H264 decoding with DCT as specialized workload.

The mean absolute error is very low at 2.6%. However, as the simulator does not capture the effects of cache contention and barrel processing, error increases along with the number of processing elements used: the maximal absolute error reaches 8%.

A First Step to Performance Prediction for Heterogeneous Processing on Manycores | DACLE Division | June 1, 2015 | 11
Evaluation on a UltraSparc T1, 1/2

A 8 core system, with 4 threads per core.
Gomet runtime assigns thread/core affinity so that the number of working threads per physical core remains as low as possible.
Workload: H264 decoding with DCT as specialized workload.

The mean absolute error is very low at 2.6 %. However, as the simulator does not capture the effects of cache contention and barrel processing, error increases along with the number of processing elements used: the maximal absolute error reaches 8 %.
Setting the number of GPPs to 8 and the number of SPs to 4
Setting the number of GPPs to 8 and the number of SPs to 4

The mean absolute error is kept very low at 2.2 %
The target architecture is an Intel Xeon W3520 Nehalem clocked at 2.67 GHz with 16 GB of RAM coupled to a NVIDIA Quadro FX 5600. The number of GPPs is set to three and the number of SPs to one.

The mean absolute error is 3.8 %.
Evaluation on a CPU+GPU

The target architecture is an Intel Xeon W3520 Nehalem clocked at 2.67 GHz with 16 GB of RAM coupled to a NVIDIA Quadro FX 5600. The number of GPPs is set to three and the number of SPs to one.

The mean absolute error is 3.8 %.
Heterogeneous computing is a hot-topic for research.

- **Offload / Sieve:** an noted source code, source to source compilation and runtime support. Not generic.
- **SkePU:** templates for parallel kernel definition. Does not allow for a collaboration between the GP cores and the SP cores.
- **Harmony:** programming framework loosely based on Out-Of-Order. More generic execution model, runtime building of the scheduling.
- **StarPU:** large size on-going research work.
Conclusion and Future Work

- Simple model but good accuracy w.r.t. real world performance
- Allows to determine the best configuration in number of jobs
- Tested of several systems but not yet embedded processors
Conclusion and Future Work

- Simple model but good accuracy w.r.t. real world performance
- Allows to determine the best configuration in number of jobs
- Tested of several systems but not yet embedded processors

- Our goal is to use this predictor in Gomet parallelizer to automatically parallelize and map heterogeneous workloads on heterogeneous targets
- Our preliminary results on CellBE processor shows promising results
Thanks!