Improving performance of parallel OpenMP programs

Artur Podobas
podobas@kth.se
SCS Seminar 2015
KTH
Overview

• Introduction

• What is OpenMP?

• My work
  – A Comparative performance study of common and popular task-centric programming frameworks
  – TurboBLYSK: Scheduling for Improved Data-driven Task Performance with Fast Dependency resolution
  – OpenMP-driven FPGA acceleration
  – Exploring Heterogeneous Scheduling using the Task-Centric Programming model
  – Architecture-aware Scheduling: a thermal approach

• Conclusion
Introduction

- Why parallel computing?
- History has judged the “golden frequency ride” to be over
  - Too much silicon / too little improvement
  - Power / Thermal limitations
- Multi-/Many-cores processors is a better alternative
  - Processors with many “weaker” cores
    - Lower frequency -> Lower Voltage -> Less Power
  - Cores work together to solve problems
    - More accurate computations (Gustafson's law)
    - Faster (Amdahl's law)
    - Run concurrently
- Multicore is a promising direction, but how do we use them?
  - MPI?
  - POSIX threads?

Image source: S. Borkar, Intel
What is OpenMP

- OpenMP is a programming model for exploiting parallelism in C/C++/Fortran applications
  - Compiler directives indicating potential parallelism
  - Easy and accomplished without disturbing the original (serial) application
- Simple to use
  - Increases portability
- First version OpenMP 1.0 in 1997
  - OpenMP Architecture Review Board
  - Started of as a thread-centric model
- Assumes Shared-Memory
  - Often together with MPI for HPC
What is OpenMP

- **OpenMP is directive-based**
  - All directives start with `#pragma omp`

- **Examples of constructs involve:**
  - Creating a number of threads
  - Parallelizing for-loops
  - Critical sections, Barriers and Atomic updates

```c
#pragma omp parallel private(i,x)
#pragma omp for reduce(+:sum)
  for (i=0; i<num_steps; i=i+1){
    x=(i+0.5)*step;
    sum = sum + 4.0/(1.0+x*x);
  }
```
What is OpenMP

• Not all parallel patterns can be exposed using the thread-centric view
  – For example traversing a list and perform work on each element in parallel

• To cope with such patterns we need a different, asynchronous, primitive

```c
while (list->next)
{
    do_work(list);
    list = list->next;
}
```
What is OpenMP: Tasks

- OpenMP 3.0 included *tasking*\(^1\)
- A task is an asynchronous parallel primitive
  - Not tied to any thread
- Directive: `#pragma omp task`
  - The statement that follows will be an asynchronous task

```c
while (list->next)
{
    do_work(list);
    list = list->next;
}
```

```c
#pragma omp parallel
#pragma omp single
while (list->next)
{
    #pragma omp task
    do_work(list);
    list = list->next;
}
#pragma omp taskwait
```

What is OpenMP: Tasks (fork/join)

- OpenMP 3.0 tasks are commonly used in a fork/join manner
  - Programmer must explicitly synchronize
- Synchronization is done with the taskwait statement
  - Blocks program execution until all previous tasks have finished executing

```c
int fib ( int n )
{
    int x, y;
    if (n<2) return n;
    #pragma omp task shared(x)
    x = fib(n-1);
    #pragma omp task shared(y)
    y = fib(n-2);
    #pragma omp taskwait
    return x+y;
}
```

Dynamically unfolding Task Graph

SCS Seminar, Artur Podobas, podobas@kth.se
What is OpenMP: Tasks (data-flow)

- OpenMP 4.0 was released 2014
  - Support for *data-flow* tasks was added
- Data-flow tasks does *not* require explicit synchronization, but...
  - ...the programmer must information concerning how and what data a task will use
  - Dependencies are drawn transparently to the programmer
  - Since data-regions are now known, acceleration using distributed memory device is now possible

Matrix Multiplication
OpenMP 3.0 (fork/join)

```c
#pragma omp parallel
#pragma omp single
for (i = 0; i < NB; i++)
  for (k = 0; k < NB; k++)
  {
    for (j = 0; j < NB; j++)
      #pragma omp task
      matmul ( A[i][k], B[k][j], C[i][j]);
      #pragma omp taskwait
  }
```

Matrix Multiplication
OpenMP 4.0 (data-flow)

```c
#pragma omp parallel
#pragma omp single
for (i = 0; i < NB; i++)
  for (k = 0; k < NB; k++)
  {
    for (j = 0; j < NB; j++)
      #pragma omp task depend(in : A[i][k], B[k][j]) depend(inout : C[i][j])
      matmul ( A[i][k], B[k][j], C[i][j]);
      #pragma omp taskwait
  }
```
What is OpenMP : Compilers

- A compiler is used to transform directives into run-system calls
- Academic/research compilers tend to be transcompilers (source-to-source compilers)
  - ROSE
  - Nanos Mercurium
- Non-academic compilers are coupled with the middle-/back-end:
  - GCC
  - Intel C Compiler
  - Sun C Compiler
  - Open64
- An alternative (and often used) way in academia is to build a run-time system API compatible with an existing compiler
What is OpenMP: Run-Time systems

- The run-time system is the heart of the OpenMP model
  - Book-keeps and manages tasks
  - Manages and Schedules tasks onto hardware resources transparently to the user
  - Draws dependencies between tasks
What is OpenMP: Run-Time System

- Task Creation
- Dependency Resolving
- Scheduler

Application | Task-Graph Dependencies | Scheduling Policy | Ready-Tasks | Hardware

- CPU0
- CPU1
- CPU2

SCS Seminar, Artur Podobas, podobas@kth.se
A Comparative performance study of common and popular task-centric programming frameworks
A Comparative performance study of common and popular task-centric programming frameworks

• Contributions:
  – A power and performance comparison of existing task-parallel run-time system
  – Explanation as to why particular run-time systems perform well while others do not

• Authors: Artur Podobas, Mats Brorsson, Karl-Filip Faxén
  – Concurrency and Computation: Practice and Experience (2013): Wiley Online Library

• Summary:
  – Evaluated BOTS for 8 different programming models and implementations
    • OpenMP: GCC, Intel, SUN's, OmpSs (BSC), OpenUH (Univ. of Houston)
    • Intel Cilk Plus and Threading Building Blocks
    • Wool (SICS)
  – Evaluation was performed on an AMD Opteron 48 core SMP system and TilePRO64
A Comparative performance study of common and popular task-centric programming frameworks

- Related work

  - Most previous studies focus on quantifying OpenMP 2.0 overheads\(^1\)\(^2\)
  
  - Evaluation of task-scheduling algorithms\(^3\) has been performed with same benchmark set
  
  - Several benchmarks suites has been designed to evaluate run-time systems, e.g. UTS\(^4\), BOTS\(^5\), StarBench\(^6\), NAS, SPEC-OMP and BARS
  
  - Most papers that introduce a new run-time system or scheduler include performance comparison

---

\(^1\) Dimakopoulos et al., “A microbenchmark study of openmp overheads under nested parallelism.”, IWOMP’08


\(^3\) Duran et al., “Evaluation of openmp task scheduling strategies”, IWOMP’08

\(^4\) Olivier et al., “Evaluating openmp 3.0 run time systems on unbalanced task graphs.”, IWOMP’10

\(^5\) Duran et al., “Barcelona openmp tasks suite: A set of benchmarks targeting the exploitation of task parallelism in openmp”, ICPP’09

A Comparative performance study of common and popular task-centric programming frameworks

```c
int fib ( int n, int depth)
{
    int x,y;
    if (n<2) return n;
    #pragma omp task shared(x) if(depth<20)
        x = fib(n-1, depth+1);
    #pragma omp task shared(y) if(depth<20)
        y = fib(n-2,depth+1);
    #pragma omp taskwait
    return x+y;
}
```

Tree Depth = 20

SCS Seminar, Artur Podobas, podobas@kth.se
A Comparative performance study of common and popular task-centric programming frameworks

- Load-balancing is crucial
  - Keep all cores occupied with important work
  - GCC's Libgomp fails to do so even for a trivial case as fibonacci
A Comparative performance study of common and popular task-centric programming frameworks

Intel OpenMP's Paraver graph for Fibonacci
A Comparative performance study of common and popular task-centric programming frameworks

GCC libgomp's Paraver graph for Fibonacci
A Comparative performance study of common and popular task-centric programming frameworks

- The difference between using a centralized queue and a distributed queue can be large:
  - Lock-contention
  - Cache performance
A Comparative performance study of common and popular task-centric programming frameworks

- Notable difference between BF and the DBF scheduler:
  - And they are using the exact same framework
A Comparative performance study of common and popular task-centric programming frameworks

![Task Creation Time Graph]

- GCC
- ICC
- OpenUH
- SunCC
- Nanos++ (BF)
- TBB
- Nanos++ (DBF)
- Wool

SCS Seminar, Artur Podobas, podobas@kth.se
A Comparative performance study of common and popular task-centric programming frameworks
A Comparative performance study of common and popular task-centric programming frameworks

• A well-performing run-time system should:
  - Use a distributed-queue algorithm (work-stealing)
    • Centralized queues works well for non-recursive patterns only
  - Well written with minimal overheads to task-creation and synchronization
    • e.g lock-free queues, cache-aligned structures, ...
  - Use a well-performing compiler
  - Contain a depth-based cutoff
**TurboBLYSK**: Scheduling for Improved Data-driven Task Performance with Fast Dependency Resolution
TurboBLYSK: Scheduling for Improved Data-driven Task
Performance with Fast Dependency Resolution

• Contribution:
  – TurboBLYSK
    • Methods used in our run-time system to make dependency resolution fast
  – A proposed new clause, dep_pattern
    • Reduces overhead by re-using earlier dependency graphs
  – An evaluation and comparison

• Authors: Artur Podobas, Vladimir Vlassov, Mats Brorsson
  – 10th International Workshop on OpenMP 2014 (best paper)

• Summary:
  – Introduces a new clause for conserving dependencies across run
  – A fast dependency analysis implementation
  – Evaluations performed using well-known OpenMP data-flow benchmarks (from BARS), where performance
  can improve on up-to two times compared to GCC or OmpSs.
**TurboBLYSK: Scheduling for Improved Data-driven Task Performance with Fast Dependency Resolution**

- **Related Work**
  - Vandierendonck et al. compared a number of implementation for solving dependencies across tasks in their run-time system SWAN\(^1\).
  - OSCAR\(^2\) have concept of analyzing task-dependencies of fork/join type of tasks offline and generating a static schedule for them to improve cache utilization.
  - OpenUH\(^3\) proposed a simpler dependency schema based on values. The re-usage scheme in our work essentially transform a dynamically detected dependency scheme to one such as their, maintaining all benefits.
  - Several programming models today supports data-flow parallelism in the task-centric framework, for example OmpSs\(^4\), LibKomp\(^5\) and OpenSTREAM\(^6\), GCC and Intel CC.

\(^1\) Vandierendonck et al., "Analysis of dependence tracking algorithms for task dataow execution.", TACO’13

\(^2\) Nakano et al., "Static coarse grain task scheduling with cache optimization using openmp”, HPC’06,

\(^3\) Chapman et al., "Support for dependency driven executions among openmp tasks.", DFM’12

\(^4\) Duran et al., "Ompss: a proposal for programming heterogeneous multi-core architectures.”, PP’11

\(^5\) Gautier et al. “Libkomp, an efficient openmp runtime system for both fork-join and data flow paradigms.”, IWOMP’12

\(^6\) Pop et al., "OpenStream: Expressiveness and data-flow compilation of OpenMP streaming programs.”, TACO’13
**TurboBLYSK: Scheduling for Improved Data-driven Task Performance with Fast Dependency Resolution**

- Gauss-Seidel application
  - Data-flow OpenMP 4.0 style
  - 2048x2048 resolution
  - Modified GCC 4.9
  - **128x128** vs **24x24** block-size (granularity)

---

**Coarse granularity (few dependencies)**

**Fine granularity (many dependencies)**
Observation: Many applications task-graphs are not input sensitive, and thus can be conserved across runs

- Use the run-time system to automatically derive dependencies
- When finished executing, label that task-graph
- Automatically optimize the information:
  - Remove unnecessary dependencies
  - Improve the mapping of locks against the dependency structure
- The next time the same task-graph is used, reapply the conserved and optimized dependency information
  - Instead of using the expensive dynamic version

**TurboBLYSK:** *Scheduling for Improved Data-driven Task Performance with Fast Dependency Resolution*
#pragma omp task depend(in : A) depend(out : B)
T1();

#pragma omp task depend(in : A,B) depend(out : C)
T2();

#pragma omp task depend(in : A,B,C) depend(out : D)
T3();

TurboBLYSK: Scheduling for Improved Data-driven Task Performance with Fast Dependency Resolution

Dynamically derive dependencies

Optimize and Rename

Static re-apply on next run

#pragma omp task depend(out : 2)
T1();

#pragma omp task depend(in : 2) depend(out : 3)
T2();

#pragma omp task depend(in : 3) depend(out : 4)
T3();

SCS Seminar, Artur Podobas, podobas@kth.se
**TurboBLYSK:** Scheduling for Improved Data-driven Task Performance with Fast Dependency Resolution

**Synthetic Benchmark - Controlled Task Granularity**

- GCC
- OmpSs
- TurboBLYSK
- TurboBLYSK (Dep_Pattern-clause)

Speedup vs. Per Task Granularity (microseconds)
**TurboBLYSK**: Scheduling for Improved Data-driven Task Performance with Fast Dependency Resolution

![Graph showing speedup vs. block size for SparseLU (2400x2400)]

- GCC
- OmpSs
- TurboBLYSK
- TurboBLYSK (Transfer-clause)
OpenMP-driven FPGA Acceleration
OpenMP-driven FPGA Acceleration

- **Contributions**
  - Methodology for using the OpenMP programming model to automatically derive an System-on-Chip capable of accelerating computations
  - HyperTasks: the hardware instantiation of a OpenMP tasks, capable of sharing resource with other tasks and using variable-width vector units

- **Authors: Artur Podobas**
  - Presented at: *IEEE 8th International Symposium on Embedded Multicore/Many-core SoCs, 2014*

- **Authors: Artur Podobas and Mats Brorsson**
  - To Be Submitted: *HyperTasks: Accelerating OpenMP computations on FPGAs*

- **Summary:**
  - Automatically constructs System-on-Chips based of a OpenMP source-code
  - The SoC have one general-purpose component (Nios2) and a number of automatically generated accelerators, which in turn can be composed of a varied number of HyperTasks
  - The performance varies between benchmarks, but can reach the equivalence of 20 AMD Opteron cores or 40 Xeon PHI cores
OpenMP-driven FPGA Acceleration

• Related work:
  
  - OmpSs have support for FPGA acceleration, either by using existing kernels\(^1\) or using Xilinx VIVADO to auto-generate code that can be synthesized code\(^2\)

  - LegUp\(^3\) is C to VHDL high-level synthesis tool that has been extended to support OpenMP 2.0 primitives (threads) on FPGAs\(^4\)

  - Leow et al.\(^5\) follows a similar concept as our, in which they convert OpenMP code to state-machines that control how data flows in the computation, and supports OpenMP 2.0 primitives

  - Durase\(^6\) and NAPA C\(^7\) are methods to integrate custom logic into soft-core to improve some parts of the application

  - SPMD (SIMT) paradigm tools exists such as Alteras OpenCL or CUDA\(^8\) to FPGA

  - Commercial tools include such as Altera’s C2H and Xilinx VIVADO

---

1 Cabrera et al., “OpenMP extensions for FPGA Accelerators”, SAMOS’09
2 Filgueras et al., “Ompss@zynq all-programmable soc ecosystem”, FPGA’14
3 Choi et al., "Legup: high-level synthesis for fpga-based processor/accelerator systems.", FPGA’11
4 Choi et al., "From software threads to parallel hardware in high-level synthesis for fpgas.", FPGA’13
5 Leow et al., “Generating hardware from openmp programs”, FPGA’06
7 Gokhale et al., "Napa c: Compiling for a hybrid risc/fpga architecture", FPGAs for Custom Computing Machines, 1998
8 Papakonstantinou et al., “FCUDA: Enabling efficient compilation of CUDA kernels onto FPGAs”, SASP’09
OpenMP-driven FPGA Acceleration

- Four reason for moving towards automatically generated FPGAs capable of running OpenMP:
  - OpenMP programmers are encouraged to expose as much parallelism as possible (but *not too much!*)
    - Necessary for load-balancing and reducing the make-span
    - But current architectures cannot scale to the abundant amount of parallelism (usually) offered by application
OpenMP-driven FPGA Acceleration
OpenMP-driven FPGA Acceleration

AMD Athlon processor die photo
OpenMP-driven FPGA Acceleration

AMD Athlon processor die photo
OpenMP-driven FPGA Acceleration

AMD Athlon processor die photo
OpenMP-driven FPGA Acceleration

AMD Athlon processor die photo
OpenMP-driven FPGA Acceleration

AMD Athlon processor die photo
OpenMP-driven FPGA Acceleration

• Four reason for moving towards automatically generated FPGAs capable of running OpenMP:
  – OpenMP programmers are encouraged to expose as much parallelism as possible (but *not too much!*)
    • Necessary for load-balancing and reducing the make-span
    • But current architectures cannot scale to the abundant amount of parallelism (usually) offered by application
  – General purpose processor contain too much “useless” logic
OpenMP-driven FPGA Acceleration

• Four reasons for moving towards automatically generated FPGAs capable of running OpenMP:
  - OpenMP programmers are encouraged to expose as much parallelism as possible (but *not too much!*)
    • Necessary for load-balancing and reducing the make-span
    • But current architectures cannot scale to the abundant amount of parallelism (usually) offered by application
  - General purpose processor contain too much “useless” logic
  - Hardware designers are becoming scarce\(^1\)
  - FPGAs can accelerate computations!

OpenMP-driven FPGA Acceleration

(a) OpenMP Source Code
void pi (int _start, int end, float *acc);
for (i=0; i<1000; i+=10)
#pragma omp task
pi (i,i+10,&result);

(b) Front-End
Lex, Parse, Syntax/Type-Check,...

(c) Intermediate-Form
... schedule_task ( t ); }

(d) Components
Multiplexers
Registers
Adders
SQRT
Comparators ...

(e) Accelerator-Generator

(f) System-Generator

(h) System-On-Chip description
QSYS / SOPC

(g) Source-to-Source
for (i=0; i<1000; i+=10)
#pragma omp task
pi (i,i+10,&result);
{ bl_task *t = new_task();
t->_start = i;
t->_end = i+10;
t->_acc = &result;
schedule_task ( t ); }

(NIOS-2 C / Headers)

QSYS / SOPC
OpenMP-driven FPGA Acceleration

PI-Kernel

![Graph 1: SpeedUp vs HyperTasks](image1)

![Graph 2: SpeedUp per kALM](image2)
OpenMP-driven FPGA Acceleration

PI-Kernel

Corresponding Performance (cores)

- AMD Opteron
- Xeon PHI
Exploring Heterogeneous Scheduling using the Task-Centric Programming Model
Exploring Heterogeneous Scheduling using the Task-Centric Programming Model

• Contributions:
  - A run-time system capable of handling the OmpSs programming model and utilize GPUs and TilePRO64 accelerator simultaneously
  - A scheduling policy that dynamically adjusts itself to the computation and is likely to schedule things where it runs fastest

• Authors: Artur Podobas, Vladimir Vlassov, Mats Brorsson
  - Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms, HeteroPAR'12

• Summary:
  - Performance evaluation of common "homogeneous" scheduling techniques on a heterogeneous platform
  - A run-time supports GPUs and TilePRO64 in co-acceleration with the hosts CPU
  - A new scheduling policy that observes trends concerning tasks on the heterogeneous system, comparing it to traditional scheduling policies
Exploring Heterogeneous Scheduling using the Task-Centric Programming Model

• Related work
  - *SSs\(^1\)\(^2\) support both GPUs and the IBM Cell processor together with all (nearly) of their schedulers
  - StarPU\(^3\) by INRIA is a accelerator-focused task run-time system that handles multiple accelerators and uses the HEFT\(^4\) algorithm
  - O’Brien\(^5\) focuses on executing OpenMP on the Cell architecture, with extra focus on software-cachine.
  - Liu et al.\(^6\) extended the OpenMP programming such that it supports their 3SOC system

\(^1\) Ayguade et al., “An extension of the starss programming model for platforms with multiple gpus”, EuroPAR’09
\(^2\) Bellens et al., “Cellss: a programming model for the cell be architecture”, SC’06
\(^3\) Augonnet et al., “StarPU: A unified platform for task scheduling on heterogeneous multicore architectures”, EuroPAR’09
\(^4\) Topcuoglu et al., “Performance-effective and low-complexity task scheduling for heterogeneous computing”, IEEE Parallel and Distributed Systems, 2002
\(^6\) Liu et al., “Extending openmp for heterogeneous chip multiprocessors”, Parallel Processing,2003
Exploring Heterogeneous Scheduling using the Task-Centric Programming Model

- Built a run-time system capable of handling multiple accelerators
  - Data-flow OpenMP model
  - GPU(s)
  - TilePRO64
- Programmer supplies different versions of the same task
- Software-caching
  - Double-buffering
- Due to TilePRO64 not supporting offloading, we wrote a micro-kernel for PCI communication
Exploring Heterogeneous Scheduling using the Task-Centric Programming Model

Random Work-Dealer

Random Work-Stealer

Weighted-Random

SCS Seminar, Artur Podobas, podobas@kth.se
Exploring Heterogeneous Scheduling using the Task-Centric Programming Model

- We introduced our FCRM scheduler
  - Correlates the performance of a task on a certain heterogeneous device to its input-set
    - Input-set is specified by the programmer and could be e.g. the size of data that the task will work with
  - During scheduler, tasks are randomly scheduled according to their "predicted" performance:
    - Factors in memory transfer overhead, i.e., software-caching
Exploring Heterogeneous Scheduling using the Task-Centric Programming Model

- **Result: Matrix Multiplication**
  - Our scheduling method, at the first run, does not perform well. Rather, the more time you use it the better it performs.
Exploring Heterogeneous Scheduling using the Task-Centric Programming Model

• Result: Incremental improvement
  – The more time you used our model, the better it becomes
Architecture-aware task-scheduling: a thermal approach
Architecture-aware task-scheduling: a thermal approach

• Contributions:
  – A thermal-aware scheduler that balances the power-consumption of cores
  – Evaluation of our scheduler against a non-aware scheduler, showing performance improvement potential

• Authors: Artur Podobas and Mats Brorsson
  – Presented at: First International Workshop on Future Architectural Support for Parallel Programming (FASPP’11)

• Summary:
  – Thermal-aware scheduler, written as a scheduler-plugin in Nanos++
  – Simulated thermal properties of a 16-core TilePRO64 processor
  – McPAT\(^1\) used for power modeling and a simplified HotSpot\(^2\) model was derived
  – Compared our scheduler against a traditional the OmpSs default and breadth-first schedulers

---

\(^1\) Li et al., "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures", MICRO’09

\(^2\) Shadron et al., "HotSpot: A compact thermal modeling methodology for early-stage VLSI design", VLSI'06
Related work:

- Online methods include ARMA\(^1\) to predict temperature, having cores executing dummy work\(^2\) to cool cores, coolest-core scheduling\(^3\), Heat-and-Run\(^4\) or clustering application according to their behavior and interleaving them\(^5\).

- Juan et al. Investigate OpenMP for-loop scheduling together with DVFS to reduce power consumption\(^6\).

- Offline scheduling techniques include formulating the problem as ILP\(^7\) and using heuristics if the computation deviates from the static properties.

\(^1\) Coskun et al. "Temperature aware task scheduling in mpsocs." DATE ’07

\(^2\) Choi et al. "Thermal-aware task scheduling at the system software level" ISLPED’07

\(^3\) Cui et al. “Dynamic thermal-aware scheduling on chip multiprocessor for soft real-time system”, GLSVLSI’09

\(^4\) Gomaa et al., “Heat-and-run: leveraging SMT and CMP to manage power density through the operating system.”, ACM SIGARCH 2004

\(^5\) Juan et al., "Energy constrained openmp static loop scheduling.", HPCC’08

\(^6\) Coskun et al., "Temperature-aware mpsoc scheduling for reducing hot spots and gradients.", ASP-DAC’08
Given enough variety of tasks with different power-consuming properties, can we schedule them such that temperature is manageable (ie does not throttle)?

Scheduler proposed:
- Measure IPC for every tasks executed. IPC correlates well with power.
- Spawn child-tasks of the profiled task into one of the many queues depending on parents estimated power-properties
- Depending on the temperature of the core, execute tasks from 'cool' or 'hot' queues
Results:

- When there is a variety of tasks, then our scheduler can improve performance (under thermal constraints).
- When tasks are more homogeneous, temperature throttling is inevitable.
Ending Conclusion

• Improving performance in OpenMP
  – I have demonstrated how to exploit performance in different sub-areas of the OpenMP tasking model
    • Thermal-aware scheduling
    • Heterogeneity
    • FPGA acceleration
    • Run-time system improvement
  – Questions?