Hardware Heterogeneous Task Scheduling for Task-based Programming Models

Xubin Tan

OpenMPCon 2018

Advisors: Carlos Álvarez, Daniel Jiménez-González
Agenda

> Background, Motivation
> Picos++ accelerated hardware runtime
> Hardware Heterogeneous task scheduling
> Results
Agenda

> Background, Motivation

> Picos++ accelerated hardware runtime

> Hardware Heterogeneous task scheduling

> Results
Moore’s Law – Commodity Microprocessor

40 Years of Microprocessor Trend Data

- Transistors (thousands)
- Single-Thread Performance (SpecINT x 10^3)
- Frequency (MHz)
- Typical Power (Watts)
- Number of Logical Cores

Multi-core, many-core, heterogeneous architectures

With task-based programming models (OpenMP, OmpSs, Codelet, StarPU, …, etc), an application can be expressed as a collection of tasks with dependences.
void multisort (size_t n, T data[n], T tmp[n]) {

Programmer  #pragma omp task inout(data[0], tmp[0])
             multisort(i, &data[0], &tmp[0]);
             ...

#pragma omp task in(data[0], data[i]) out(tmp[0])
  merge(i, &data[0], &data[i], &tmp[0]);
             ...
}

Compiler (Mercurium)  generates runtime calls to create tasks (task descriptor)

Runtime (Nanos++) manages task creation, constructs task dependence graph, and schedules ready tasks dynamically
SW runtime for exploiting fine-grained task parallelism

Smaller tasks, more tasks, more parallelism

Four OmpSs applications with problem size 2K*2K, with 12 cores
Use hardware to accelerate SW runtime

Finer tasks, more tasks, more parallelism

Four OmpSs applications with problem size 2K*2K, with 12 cores
Costly task-dependence analysis and heterogeneous task scheduling

New task  Ready task  Finish task
Agenda

> Background, Motivation
> Picos++ accelerated hardware runtime
> Hardware Heterogeneous task scheduling
> Results
Picos++ System

Picos++, several HwAccs @ different frequency

Two types of data comm.
Agenda

> Background, Motivation
> Picos++ accelerated hardware runtime
> Hardware Heterogeneous task scheduling
> Results
Heterogeneous Task Scheduling

Two main issues concerning scheduling:

A. Load imbalance
   Scheduling task to a suitable hardware device with the least number of waiting tasks

B. Hardware to hardware communication
   Picos++ directly manages task scheduling to different HW units

Related work:
A. Static algorithms
   Heterogeneous Earliest Finish Time (HEFT), Critical-Path-on-a-Processor (CPOP), etc

B. Dynamic algorithms
   Criticality-aware task scheduler (CATS), Critical-path scheduler (CPATH), Version Scheduler from Judit Planas, etc

C. Scheduling hierarchies
   Intel CARBON, Asynchronous Direct Messages (ADM), Task Scheduling Unit, Programmable Task Management Unit (TMU), etc
Picos++ System

Picos++, several HwAccs @ different frequency

Two types of data comm.
> Ready tasks come from Picos or Bypass

> Hardware Registers for each device to keep information

> Device selection based on:
  - Target device of the task
  - Hardware Register information about the number of pending tasks for each device
  - HW accelerator priority
Agenda

> Background, Motivation
> Picos++ accelerated hardware runtime
> Hardware Heterogeneous task scheduling
> Results
Experimental Setup

A Xilinx Ultrascale+ MPSoC:
- 4 ARM cores @ 1.1GHz, and a FPGA
- HW registers for power samples
- OmpSs up running in Linux 16.04

The SW-only runtime @ 1.1GHz is supported by OmpSs @ FPGA, Picos++ @ 100 or 200MHz
An example code of gemm function for Cholesky

```c
#pragma omp target device(fpga) copy_deps onto(0) num_instances(4)
#pragma omp task inout([bs]C) in([bs]A, [bs]B)

void matmulBlock(T (*A)[bs], T (*B)[bs], T (*C)[bs]){
  unsigned int i, j, k;

  #pragma HLS array_partition variable=A block factor=bs/2 dim=2
  #pragma HLS array_partition variable=B block factor=bs/2 dim=1
  for (i = 0; i < bs; i++) {
    for (j = 0; j < bs; j++) {
      #pragma HLS pipeline II=1
      T sum = 0;
      for (k = 0; k < bs; k++) {
        sum += A[i][k] * B[k][j];
      }
      C[i][j] += sum;
    }
  }
}

#pragma omp target device(smp) no_copy_deps implements(matmulBlock)
#pragma omp task in([bs]A, [bs]B) inout([bs]C)

void matmulBlockSmp(T (*A)[bs], T (*B)[bs], T (*C)[bs]){
  T const alpha = 1.0; T const beta = 1.0;
  cblas_gemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
             bs, bsize, bs, alpha, a, bs, b, bs, beta, c, bs);
}
```
OmpSs@FPGA | Compilation process

- C source code
- Mercurium (fpgacc, fpgacxx)
- autoVivado
- Native compiler
- Linker
- BIT
- EXE
- FPGA specific vendor tools

Tools:
- nanos
- TASKs
- MA
- Picos
Task scheduling using 4 different HwAccs-only

Vs Seq execution in SMP:
1.7x speedup,
0% energy savings

Vs SW-only runtime with similar HW
1.1x speedup
18% energy savings

The number of tasks executed in this hardware device

SMP threads

SMP is only used for task creation, not for execution
Task scheduling using 4 different HwAccs+SMP

Vs Seq execution in SMP:
  3x speedup,
  42% energy savings
Vs SW-only runtime with similar HW
  1.95x speedup
  53% energy savings

Picos++ schedules nearly half of GEMM tasks to SMP
Task scheduling using 4 same HwAccs+SMP

Vs Seq execution in SMP:
- 6.5x speedup,
- 73% energy savings

Vs SW-only runtime with similar HW:
- 1.36x speedup
- 33% energy savings

Use HwAccs only for GEMM task execution, further balance the workloads:
- HwAcc3 has the highest priority, SMP has the lowest
Performance Impact of the Task Granularity

With 4 threads

Picos++ vs Seq: up to 3.4x speedup, 65% energy saving
vs SW-only: up to 1.6x speedup, 40% energy saving
Performance Impact of the Heterogeneous Task Management with SMP+4 HwAccs*

Matmul:
Vs seq: 11.2x, 70% of energy savings
Vs SW-only: 2.7x, 65% of energy savings

Multisort:
Vs seq: 5.9x, 85% of energy savings
Vs SW-only: 2.7x, 69% of energy savings

with SMP+4 HwAccs*
Scaling Up the Number of HwAccs

Matmul (2k, 32) with Picos++ at 100 or 200 MHz
By using Picos++ runtime, the cost of FPGA task management is significantly smaller.
Paraver Trace of Cholesky@SMP+4GEMM Accs

Task Instances

main

omp_potrf

omp_trsm

omp_syrk

Picos API Activities

Send new task

Send new task
Polling for ready/executed Hw tasks
Send finish task
### Potential Time Saving During Cholesky Execution

#### TABLE 7: The useful time consumption in a 2K*2K trace

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>T1/Hw0*</th>
<th>T2/Hw1*</th>
<th>T3/Hw2*</th>
<th>T4/Hw3*</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tasks</strong></td>
<td>main</td>
<td>97%</td>
<td>1.24%</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>omp_potrf</td>
<td>0.14%</td>
<td>0.78%</td>
<td>0.76%</td>
<td>0.75%</td>
</tr>
<tr>
<td></td>
<td>omp_syrk</td>
<td>0.28%</td>
<td>15.32%</td>
<td>14.47%</td>
<td>15.00%</td>
</tr>
<tr>
<td></td>
<td>omp_trsm</td>
<td>0.26%</td>
<td>20.79%</td>
<td>20.79%</td>
<td>22.64%</td>
</tr>
<tr>
<td></td>
<td>omp_gemm*</td>
<td>13.34%</td>
<td>56.52%</td>
<td>56.52%</td>
<td>60.89%</td>
</tr>
<tr>
<td><strong>Picos++APIs</strong></td>
<td>New task</td>
<td>33.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Polling ready</td>
<td>0.81%</td>
<td>19.19%</td>
<td>19.16%</td>
<td>18.49%</td>
</tr>
<tr>
<td></td>
<td>Successful ready</td>
<td>0.05%</td>
<td>5.89%</td>
<td>5.52%</td>
<td>4.78%</td>
</tr>
<tr>
<td></td>
<td>Finish task</td>
<td>0.04%</td>
<td>3.08%</td>
<td>3.07%</td>
<td>3.06%</td>
</tr>
</tbody>
</table>

**Potential time candidate for reduction**

| HwAccs Threads   | omp_gemm*          | 86.66%  | 43.48%  | 43.48%  | 39.11%  |
| Upper bound      | 0                 | 48.63%  | 54.75%  | 53.11%  |
GFLOPS: Picos++ with HPC

Matmul (2K, 128), Picos++@100MHz, HwAccs @ 300MHz

With more resources, Picos++ gains more GFLOPS
Up to 49 GFLOPS, 24% faster than SW-only

5.74 watts average power consumption
**GFLOPS: Picos++ with HPC**

Matmul (2K, 128), Picos++ @ 100MHz, HwAccs @ 300MHz

<table>
<thead>
<tr>
<th>Name</th>
<th>Gflops per watt</th>
<th>Num.Threads, Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel(R) Core(TM) i5-3470</td>
<td>0.51</td>
<td>4t, 3GHz</td>
</tr>
<tr>
<td>Intel(R) Xeon(R) CPU E5-2020 V2</td>
<td>4.14</td>
<td>24t, 2.1GHz</td>
</tr>
<tr>
<td>Intel(R) Core(TM) i7-4600U CPU</td>
<td>4.75</td>
<td>4t, 2.1GHz</td>
</tr>
<tr>
<td>Picos++ with 3 blocksize 128 Accs</td>
<td>8.53</td>
<td>4t, 1.1GHz</td>
</tr>
</tbody>
</table>
➢ Fine-grained parallelism offers a lot of opportunities for desirable performance at a low energy cost

➢ HW task-dependence manager and heterogeneous task scheduler are fast, energy efficient, and general purpose

➢ The more hardware resources, the greater the impact of Picos hardware

➢ Picos++ is compatible for task-based programming model runtimes as gomp for OpenMP
Jaume Bosch, Xubin Tan, Jaume Bosch, Antonio Filgueras, Miquel Vidal, Marc Mateu, Daniel Jiménez-González, Carlos Álvarez, Xavier Martorell, Eduard Ayguadé, Jesus Labarta. “Application Acceleration on FPGAs with OmPss@FPGA” in FPT 2018.


Jaume Bosch, Xubin Tan, Carlos Álvarez, Daniel Jiménez-González, Eduard Ayguadé, Mateo Valero. “Characterizing and Improving the Performance of Many-Core Task-Based Parallel Programming Runtimes” in IPDPS Workshop IPDRM’17.

This work has been financially supported by the Spanish Government through Programa Severo Ochoa (SEV-2015-0493), by the Spanish Ministry of Science and Technology through TIN2015-65316-P project, by the Generalitat de Catalunya (contracts 2014-SGR-1051 and 2014-SGR-1272), by the European Research Council RoMoL Grant Agreement number 321253 and by the OmpSs on Android and Hardware support for runtime Project Cooperation Agreement with LG Electronics. We also thank the Xilinx University Program for its hardware and software donations.
Hardware Heterogeneous Task Scheduling for Task-based Programming Models

Xubin Tan

OpenMPCon 2018

Advisors: Carlos Álvarez, Daniel Jiménez-González
HW and Power cost in a Xilinx Ultrascale+ MPSoC

TABLE 3: Hardware Resource and Power Consumption

<table>
<thead>
<tr>
<th>Name</th>
<th>FPGA resource</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BRAM18Kb</td>
<td>DSP48E</td>
</tr>
<tr>
<td>XCZU9EG</td>
<td>1824</td>
<td>2520</td>
</tr>
<tr>
<td>Picos++</td>
<td>87/5.0%</td>
<td>0</td>
</tr>
<tr>
<td>Comm. Logic</td>
<td>2/0%</td>
<td>0</td>
</tr>
<tr>
<td>APU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 ARM Cortex-A53s</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
# TABLE 1: The characteristics of real benchmarks

<table>
<thead>
<tr>
<th>Name</th>
<th>Configs</th>
<th>#Tasks</th>
<th>Seq time(us)</th>
<th>Latency(us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matmul</td>
<td>(2K, 32)</td>
<td>262144</td>
<td>6820850</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>(2K, 64)</td>
<td>32768</td>
<td>5463956</td>
<td>167</td>
</tr>
<tr>
<td></td>
<td>(2K, 128)</td>
<td>4096</td>
<td>5435528</td>
<td>1327</td>
</tr>
<tr>
<td>Cholesky</td>
<td>(2K, 32)</td>
<td>45760</td>
<td>1232664</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>(2K, 64)</td>
<td>5984</td>
<td>1087485</td>
<td>182</td>
</tr>
<tr>
<td>Multisort</td>
<td>(1M, 256, 256k)</td>
<td>9565</td>
<td>395594</td>
<td>414</td>
</tr>
<tr>
<td></td>
<td>(2M, 512, 512K)</td>
<td>9565</td>
<td>832882</td>
<td>871</td>
</tr>
<tr>
<td></td>
<td>(1M, 1K, 512K)</td>
<td>2397</td>
<td>407648</td>
<td>1701</td>
</tr>
</tbody>
</table>
### TABLE 2: Characteristics of HwAccs in XCZU9EG-FFVC900

<table>
<thead>
<tr>
<th>Name</th>
<th>HWACCs</th>
<th>Latency (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B_18Kb</td>
<td>DSP48E</td>
</tr>
<tr>
<td>fgemm32</td>
<td>68/3.7%</td>
<td>160/6.4%</td>
</tr>
<tr>
<td>fsyrrk32</td>
<td>36/2.0%</td>
<td>160/6.4%</td>
</tr>
<tr>
<td>ftrsm32</td>
<td>36/2.0%</td>
<td>104/4.1%</td>
</tr>
<tr>
<td>fpotrf32</td>
<td>10/0.6%</td>
<td>22/0.9%</td>
</tr>
<tr>
<td>fgemm64</td>
<td>74/4.1%</td>
<td>160/6.4%</td>
</tr>
<tr>
<td>fsyrrk64</td>
<td>42/2.3%</td>
<td>160/6.4%</td>
</tr>
<tr>
<td>ftrsm64</td>
<td>42/2.3%</td>
<td>250/9.9%</td>
</tr>
<tr>
<td>fpotrf64</td>
<td>28/1.5%</td>
<td>22/0.9%</td>
</tr>
<tr>
<td>fmatmul32</td>
<td>68/3.7%</td>
<td>162/6.4%</td>
</tr>
<tr>
<td>fmatmul64</td>
<td>138/7.6%</td>
<td>322/12.8%</td>
</tr>
<tr>
<td>fmatmul128</td>
<td>287/15.7%</td>
<td>642/25.5%</td>
</tr>
<tr>
<td>sort256</td>
<td>68/3.7%</td>
<td>0</td>
</tr>
<tr>
<td>sort512</td>
<td>138/7.6%</td>
<td>0</td>
</tr>
<tr>
<td>sort1024</td>
<td>159/8.7%</td>
<td>0</td>
</tr>
</tbody>
</table>