Steady-state scheduling on CELL

Mathias Jacquelin,

joint work with Matthieu Gallet, Loris Marchal and Yves Robert

INRIA GRAAL project-team
LIP (ENS-Lyon, CNRS, INRIA)
École Normale Supérieure de Lyon, France

“Scheduling for large-scale systems” workshop,
Outline

Introduction
  Steady-state scheduling
  CELL

Platform and Application Modeling

Mapping the Application

Practical Steady-State on CELL
  Preprocessing of the schedule
  State machine of the application
  Preliminary results

Conclusion and Future works
Motivation

- Multicore architectures: new opportunity to test the scheduling strategies designed in the GRAAL team.
- Our trademark: efficient scheduling on heterogeneous platforms
  - Most multicore architecture are homogeneous, regular
    - Need for tailored algorithms (linear algebra, ...)
  - Emerging heterogeneous multicore:
    - Dedicated processing units on GPUs
    - Mixed system: processor + accelerator
  - This study: steady-state scheduling on CELL (bounded heterogeneity) to demonstrate the usefulness of complex (static) scheduling techniques
  - Ongoing work: only preliminary results
Motivation

- Multicore architectures: new opportunity to test the scheduling strategies designed in the GRAAL team.
- Our trademark: efficient scheduling on heterogeneous platforms
- Most multicore architecture are homogeneous, regular
  - Need for tailored algorithms (linear algebra,\ldots)
- Emerging heterogeneous multicore:
  - Dedicated processing units on GPUs
  - Mixed system: processor + accelerator
- This study: steady-state scheduling on CELL (bounded heterogeneity) to demonstrate the usefulness of complex (static) scheduling techniques
- Ongoing work: only preliminary results
Motivation

- Multicore architectures: new opportunity to test the scheduling strategies designed in the GRAAL team.
- Our trademark: efficient scheduling on heterogeneous platforms
- Most multicore architecture are homogeneous, regular
  - Need for tailored algorithms (linear algebra, ...)
- Emerging heterogeneous multicore:
  - Dedicated processing units on GPUs
  - Mixed system: processor + accelerator
- This study: steady-state scheduling on CELL (bounded heterogeneity) to demonstrate the usefulness of complex (static) scheduling techniques
- Ongoing work: only preliminary results
Rationale:

- A pipelined application:
  - Simple chain
  - More complex application (Directed Acyclic Graph)

- Objective: optimize the throughput of the application (number of input files treated per second)

- Today: simple case where each task has to be mapped on one single resource
Introduction: Steady-state Scheduling

Rationale:

- A pipelined application:
  - Simple chain
  - More complex application (Directed Acyclic Graph)
- Objective: optimize the throughput of the application (number of input files treated per second)
- Today: simple case where each task has to be mapped on one single resource
Introduction: Steady-state Scheduling

Rationale:

- A pipelined application:
  - Simple chain
  - More complex application (Directed Acyclic Graph)

- Objective: optimize the throughput of the application (number of input files treated per second)

- Today: simple case where each task has to be mapped on one single resource
Introduction: Steady-state Scheduling

Rationale:
- A pipelined application:
  - Simple chain
  - More complex application (Directed Acyclic Graph)
- Objective: optimize the throughput of the application (number of input files treated per second)
- Today: simple case where each task has to be mapped on one single resource
Introduction: Steady-state Scheduling

Rationale:

- A pipelined application:
  - Simple chain
  - More complex application (Directed Acyclic Graph)

- Objective: optimize the throughput of the application (number of input files treated per seconds)

- Today: simple case where each task has to be mapped on one single resource
CELL brief introduction

- Multicore heterogeneous processor
- Accelerator extension to Power architecture
CELL brief introduction

- Multicore heterogeneous processor
- Accelerator extension to Power architecture
CELL brief introduction

- Multicore heterogeneous processor
- Accelerator extension to Power architecture

1 PPE core
- VMX unit
- L1, L2 cache
- 2 way SMT
CELL brief introduction

- Multicore heterogeneous processor
- Accelerator extension to Power architecture

- 8 SPEs
  - 128-bit SIMD instruction set
  - Local store 256KB
  - Dedicated Asynchronous DMA engine
CELL brief introduction

- Multicore heterogeneous processor
- Accelerator extension to Power architecture
CELL brief introduction

- Multicore heterogeneous processor
- Accelerator extension to Power architecture

- Element Interconnect Bus (EIB)
  - 200 GB/s bandwidth
CELL brief introduction

- Multicore heterogeneous processor
- Accelerator extension to Power architecture

- 25 GB/s bandwidth
Outline

Introduction
  Steady-state scheduling
  CELL

Platform and Application Modeling

Mapping the Application

Practical Steady-State on CELL
  Preprocessing of the schedule
  State machine of the application
  Preliminary results

Conclusion and Future works
Platform modeling

Simple CELL modeling:

- 1 PPE and 8 SPE: 9 processing elements $P_1, \ldots, P_9$, with unrelated speed,
- Each processing element access the communication bus with a (bidirectional) bandwidth $b = (25\, \text{GB/s})$,
- The bus is able to route all concurrent communications without contention (in a first step),
- Due to the limited size of the DMA stack on each SPE:
  - Each SPE can perform at most 16 simultaneous DMA operations,
  - The PPE can perform at most 8 simultaneous DMA operations to/from a given SPE.
- Linear cost communication model: a data of size $S$ is sent/received in time $S/b$
Application modeling

Application is described by a directed acyclic graph:

- Tasks $T_1, \ldots, T_n$
- Processing time of task $T_k$ on $P_i$ is $t_i(k)$,
- If there is a dependency $T_k \rightarrow T_l$, data$_{k,l}$ is the size of the file produced by $T_k$ and needed by $T_l$,
- If $T_k$ is an input task, it reads read$_k$ bytes from main memory,
- If $T_k$ is an output task, it writes write$_k$ bytes to main memory,
Application modeling

Application is described by a directed acyclic graph:

- Tasks $T_1, \ldots, T_n$
- Processing time of task $T_k$ on $P_i$ is $t_i(k)$,
- If there is a dependency $T_k \rightarrow T_l$, $\text{data}_{k,l}$ is the size of the file produced by $T_k$ and needed by $T_l$,
- If $T_k$ is an input task, it reads $\text{read}_k$ bytes from main memory,
- If $T_k$ is an output task, it writes $\text{write}_k$ bytes to main memory,
Application modeling

Application is described by a directed acyclic graph:

- Tasks $T_1, \ldots, T_n$
- Processing time of task $T_k$ on $P_i$ is $t_i(k)$,
- If there is a dependency $T_k \rightarrow T_l$, data$_{k,l}$ is the size of the file produced by $T_k$ and needed by $T_l$,
- If $T_k$ is an input task, it reads read$_k$ bytes from main memory,
- If $T_k$ is an output task, it writes write$_k$ bytes to main memory,
Target application: vocoder
Outline

Introduction
Steady-state scheduling
CELL

Platform and Application Modeling

Mapping the Application

Practical Steady-State on CELL
  Preprocessing of the schedule
  State machine of the application
  Preliminary results

Conclusion and Future works
How to compute an optimal mapping

- Objective: maximize throughput $\rho$
- Method: write a linear program gathering constraints on the mapping
- Binary variables: $\alpha_{ki} = \begin{cases} 1 & \text{if } T_k \text{ is mapped on } P_i \\ 0 & \text{otherwise} \end{cases}$
- Other useful binary variables: $\beta_{ki,j} = 1$ iff file $T_k \rightarrow T_l$ is transferred from $P_i$ to $P_j$
On the application structure:

- Each task is mapped on a processor:
  \[ \forall T_k \sum_i \alpha_i^k = 1 \]

- Given a dependency \( T_k \rightarrow T_l \), the processor computing \( T_l \) must receive the corresponding file:
  \[ \forall (k, l) \in E, \forall P_j, \sum_i \beta_{i,j}^{k,l} \geq \alpha_j^l \]

- Given a dependency \( T_k \rightarrow T_l \), only the processor computing \( T_k \) can send the corresponding file:
  \[ \forall (k, l) \in E, \forall P_i, \sum_j \beta_{i,j}^{k,l} \leq \alpha_i^k \]
Constraints 2/2

On the achievable throughput $\rho = 1/T$:

- On a given processor, all tasks must be completed within $T$:
  \[ \forall P_i, \sum_k \alpha_i^k \times t_i(k) \leq T \]

- All incoming communications must be completed within $T$:
  \[ \forall P_j, \frac{1}{b} \left( \sum_k \alpha_j^k \times \text{read}_k + \sum_{k,l} \sum_i \beta_{i,j}^{k,l} \times \text{data}_{k,l} \right) \leq T \]

- All outgoing communications must be completed within $T$:
  \[ \forall P_i, \frac{1}{b} \left( \sum_k \alpha_i^k \times \text{write}_k + \sum_{k,l} \sum_i \beta_{i,j}^{k,l} \times \text{data}_{k,l} \right) \leq T \]

+ constraints on the number of incoming/outgoing communications to respect the DMA requirements
+ constraints on the available memory on SPE
Optimal mapping computation

- Linear program with the objective of minimizing $T$
- Integer (binary) variables: Mixed Integer Programming
- NP-complete problem

- Efficient solvers exist with short running time
  - for small-size problems
  - or when an approximate solution is searched

- We use CPLEX, and look for an approximate solution (5% of the optimal throughput is good enough)
Outline

Introduction
  Steady-state scheduling
  CELL

Platform and Application Modeling

Mapping the Application

Practical Steady-State on CELL
  Preprocessing of the schedule
  State machine of the application
  Preliminary results

Conclusion and Future works
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.

- \( \text{min\_period}_l = \max_{m \in \text{precl}} (\text{min\_period}_m) + \text{peek}_l + 2 \)
- \( \text{min\_buff}_{i,l} = \text{min\_period}_l - \text{min\_period}_i \)
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.

\[ \text{min}\ buff_{i,k} = 4 \]
\[ \text{peek}_i = 0 \]
\[ \text{min}\ period_i = 0 \]
\[ \text{min}\ period_j = 3 \]
\[ \text{peek}_j = 1 \]
\[ \text{min}\ period_j = 3 \]
\[ \text{min}\ buff_{i,j} = 3 \]
\[ \text{min}\ buff_{i,l} = 9 \]
\[ \text{min}\ buff_{j,l} = 6 \]
\[ \text{peek}_l = 2 \]
\[ \text{min}\ period_l = 9 \]
\[ \text{peek}_k = 3 \]
\[ \text{min}\ buffer_{k,l} = 4 \]
\[ \text{min}\ buffer_{i,k} = 5 \]
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.

- **$T_i$**
  - Period: $1$
  - $peek_i = 0$
  - $min\_period_i = 0$
  - $min\_buff_i,k = 5$

- **$T_j$**
  - $peek_j = 1$
  - $min\_period_j = 3$
  - $min\_buff_j,l = 9$
  - $min\_buff_j,l = 6$

- **$T_k$**
  - $peek_k = 3$
  - $min\_period_k = 5$
  - $min\_buff_k,l = 4$

- **$T_l$**
  - $peek_l = 2$
  - $min\_period_l = 9$
  - $min\_buff_l,l = 9$
Main Objective: Compute minimal starting period and buffer sizes.

- $\text{period} = 2$
- $\text{peek}_i = 0$
- $\text{min\_period}_i = 0$
- $\text{min\_buff}_{i,k} = 5$
- $\text{min\_buff}_{i,j} = 3$
- $\text{min\_buff}_{i,l} = 9$
- $\text{peek}_j = 1$
- $\text{min\_period}_j = 3$
- $\text{min\_buff}_{j,l} = 6$
- $\text{peek}_k = 3$
- $\text{min\_period}_k = 5$
- $\text{min\_buff}_{k,l} = 4$
- $\text{peek}_l = 2$
- $\text{min\_period}_l = 9$
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.

period = 3

$T_i$

peek$_i = 0$
min_period$_i = 0$

min_buff$_i,k = 5$

$T_j$

peek$_j = 1$
min_period$_j = 3$

min_buff$_i,j = 3$

$T_k$

peek$_k = 3$
min_period$_k = 5$

min_buff$_j,l = 6$

$T_l$

peek$_l = 2$
min_period$_l = 9$

min_buff$_j,l = 6$
Preprocessing of the schedule

**Main Objective:** Compute minimal starting period and buffer sizes.

\[
\begin{align*}
\text{period} &= 4 \\
\text{peek}_i &= 0 \\
\text{min}_\text{period}_i &= 0 \\
\text{peek}_j &= 1 \\
\text{min}_\text{period}_j &= 3 \\
\text{peek}_k &= 3 \\
\text{min}_\text{period}_k &= 5 \\
\text{peek}_l &= 2 \\
\text{min}_\text{period}_l &= 9 \\
\text{min}_\text{buff}_{i,k} &= 5 \\
\text{min}_\text{buff}_{i,j} &= 3 \\
\text{min}_\text{buff}_{j,l} &= 6 \\
\text{min}_\text{buff}_{k,l} &= 4 \\
\end{align*}
\]
Main Objective: Compute minimal starting period and buffer sizes.

period = 5

peek_i = 0
min_period_i = 0

min_buff_i,k = 5

peek_j = 1
min_period_j = 3

min_buff_i,l = 9

peek_l = 2
min_period_l = 9

peek_k = 3
min_period_k = 5

min_buff_k,l = 4

min_buff_j,l = 6

min_buff_j,l = 3
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.

- $T_i$: $\text{period} = 6$, $\text{peek}_i = 0$, $\text{min\_period}_i = 0$, $\text{min\_buff}_{i,k} = 5$
- $T_j$: $\text{peek}_j = 1$, $\text{min\_period}_j = 3$, $\text{min\_buff}_{i,j} = 3$, $\text{min\_buff}_{i,l} = 9$
- $T_k$: $\text{peek}_k = 3$, $\text{min\_period}_k = 5$, $\text{min\_buff}_{j,k} = 6$, $\text{min\_buff}_{k,l} = 4$
- $T_l$: $\text{peek}_l = 2$, $\text{min\_period}_l = 9$, $\text{min\_buff}_{l,k} = 3$
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.

\[
\begin{align*}
\text{period} &= 7 \\
T_i &\quad \text{peek}_i = 0 \\
&\quad \text{min\_period}_i = 0 \\
\text{min\_period}_j &= 3 \\
T_j &\quad \text{peek}_j = 1 \\
&\quad \text{min\_period}_j = 3 \\
\text{min\_buff}_{i,k} &= 5 \\
\text{peek}_k &= 3 \\
&\quad \text{min\_period}_k = 5 \\
\text{min\_buff}_{j,l} &= 6 \\
\text{min\_buff}_{i,l} &= 9 \\
\text{min\_buff}_{j,i} &= 6 \\
\text{min\_buff}_{i,l} &= 9 \\
T_l &\quad \text{peek}_l = 2 \\
&\quad \text{min\_period}_l = 9 \\
\end{align*}
\]
Main Objective: Compute minimal starting period and buffer sizes.
Main Objective: Compute minimal starting period and buffer sizes.

period = 9

$T_i$
- $peek_i = 0$
- $min\_period_i = 0$
- $min\_buff_i,k = 5$
- $min\_buff_i,j = 3$
- $min\_buff_i,l = 9$

$T_j$
- $peek_j = 1$
- $min\_period_j = 3$
- $min\_buff_j,k = 4$
- $min\_buff_j,l = 6$

$T_k$
- $peek_k = 3$
- $min\_period_k = 5$
- $min\_buff_k,l = 4$

$T_l$
- $peek_l = 2$
- $min\_period_l = 9$
- $T_i = 9$
State machine of the application

Two main phases: Computation and Communication

- Select a Task
- Wait Resources
- Process Task
- Signal new Data
- Communicate
State machine of the application

Two main phases: Computation and Communication
State machine of the application

Two main phases: Computation and Communication
State machine of the application

Two main phases: **Computation and Communication**
State machine of the application

Two main phases: Computation and Communication
State machine of the application

Two main phases: Computation and Communication

- Compute
  - For each inbound comm.
    - Watch DMA
    - Check input data
    - Check input buffers
    - Get Data

- Communication Phase
State machine of the application

Two main phases: Computation and Communication

- **Compute**
  - For each inbound comm.
  - Watch DMA
  - Check input data
  - Check input buffers
  - Get Data

- **Communication Phase**
  - No more comm.
State machine of the application

Two main phases: Computation and Communication

- Compute
  - For each inbound comm.
    - Watch DMA
      - Check input data
        - Check input buffers
          - Get Data
State machine of the application

Two main phases: Computation and Communication
State machine of the application

Two main phases: Computation and Communication
State machine of the application

Two main phases: Computation and Communication

- **Compute**
  - For each inbound comm.
  - Watch DMA
  - Check input data
  - Check input buffers
  - Get Data

- **Communication Phase**
  - No more comm.
Communication between processors

\[ P_K \]

\[ T_1^{(i)} \]

\[ T_1^{(i+1)} \]

\[ P_L \]

\[ T_2^{(i-1)} \]

\[ T_2^{(i)} \]
Communication between processors

\[ P_K \]
\[ T_1^{(i)} \]
\[ T_1^{(i+1)} \]

\[ P_L \]
\[ T_2^{(i-1)} \]
\[ T_2^{(i)} \]

Signal Data(\(i\))

mfc_putb for SPES’ outbound communications.
spe_mfcio_getb for PPEs’ outbound communications to SPES.
memcpy for PPEs’ outbound communications to main memory.
Communication between processors

$mfc\_putb$ for SPEs’ outbound communications.
$spe\_mfcio\_getb$ for PPEs’ outbound communications to SPEs.
$memcpy$ for PPEs’ outbound communications to main memory.
Communication between processors

\[ P_K \]

Output buffer containing \( i \) cannot be overwritten

\[ T_1^{(i)} \]

\[ T_1^{(i+1)} \]

\[ P_L \]

Input buffers are available to store data

\[ T_2^{(i-1)} \]

\[ T_2^{(i)} \]

Signal Data(\( i \))

Get Data(\( i \))

mfc_get for SPEs’ inbound communications.

spe_mfcio_put for PPEs’ inbound communications from SPEs.

memcpy for PPEs’ inbound communications from main memory.
mfc_get for SPEs’ inbound communications.
spe_mfcio_put for PPEs’ inbound communications from SPEs.
memcpy for PPEs’ inbound communications from main memory.
Communication between processors

\[ P_K \]

\[ T_1^{(i)} \]

Output buffer containing \( i \)
cannot be overwritten

\[ T_1^{(i+1)} \]

Signal Data\((i)\)

Get Data\((i)\)

Transfer Done\((i)\)

\[ P_L \]

\[ T_2^{(i-1)} \]

\[ T_2^{(i)} \]

Input buffers are available to store data

\textbf{mfc\_putb} for SPEs’ acknowledgements.

\textbf{spe\_mfcio\_getb} for PPEs’ acknowledgements to SPEs.

Self acknowledgement of PPEs’ transfers from main memory.
Communication between processors

$mfc_{\text{putb}}$ for SPEs’ acknowledgements.

$\text{spe}\_mfcio\_getb$ for PPEs’ acknowledgements to SPEs.

Self acknowledgement of PPEs’ transfers from main memory.
Communication between processors

- **Output buffer containing** \( i \) **cannot be overwritten**
- **Output buffer containing** \( i \) **can now be overwritten**

**mfc_putb** for SPEs’ acknowledgements.

**spe_mfcio_getb** for PPEs’ acknowledgements to SPEs.

**Self acknowledgement** of PPEs’ transfers from main memory.
Preliminary results

We outperform both greedy heuristic and sequential version.

Results are obtained over 70000 periods
Outline

Introduction
  Steady-state scheduling
  CELL

Platform and Application Modeling

Mapping the Application

Practical Steady-State on CELL
  Preprocessing of the schedule
  State machine of the application
  Preliminary results

Conclusion and Future works
Feedback on Cell programming

- Multilevel heterogeneity:
  - 32 bits SPEs vs 64 bits PPE architectures
  - Different communication mechanism and constraints

- Non trivial initialization phase
  - Varying data structure sizes (32/64bits)
  - Runtime memory allocation
On-going and Future work

- Various code optimizations
  - SIMD code for SPEs
  - Reduce control overhead

- Better communication modeling
  - Is linear cost model relevant?
  - Contention on concurrent DMA operations?

- Larger platforms
  - Using multiple CELL processors
  - CELL + other type of processing units?
  - Work on communication modeling

- Design scheduling heuristics
  - MIP is costly