

Optimizing Explicit Data Transfers for Data Parallel Applications on Heterogeneous Multi-core Platforms

S. Saidi^{1,2} P.Tendulkar¹ T. Lepley² O. Maler¹



Hipeac 2012

Outline

- 1 Introduction
- 2 Optimal Granularity
 - One Processor
 - Multiple Processors
- 3 Shared Data Transfers
- 4 Experiments on the CELL Architecture



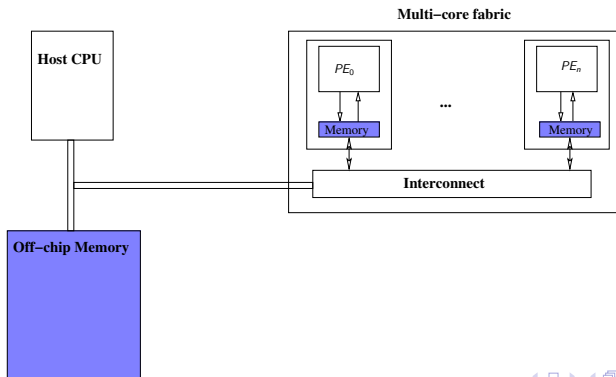
Outline

- 1 Introduction
- 2 Optimal Granularity
 - One Processor
 - Multiple Processors
- 3 Shared Data Transfers
- 4 Experiments on the CELL Architecture



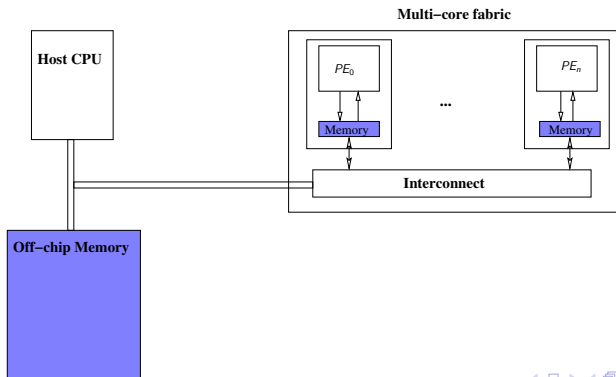
Motivation

- How to **reduce/hide** the **off-chip memory latency**?



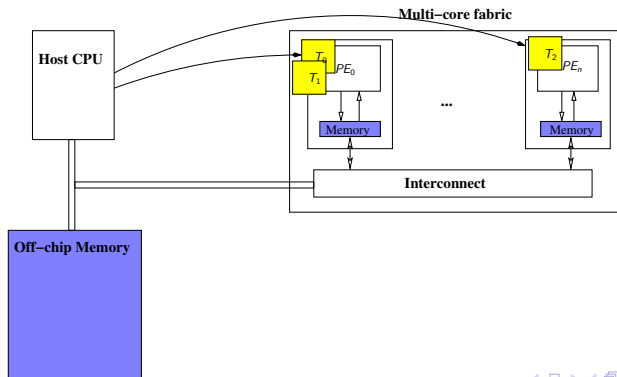
Heterogeneous Multi-core Architectures

- a powerful host processor and a multi-core fabric to **accelerate** computationally heavy kernels.



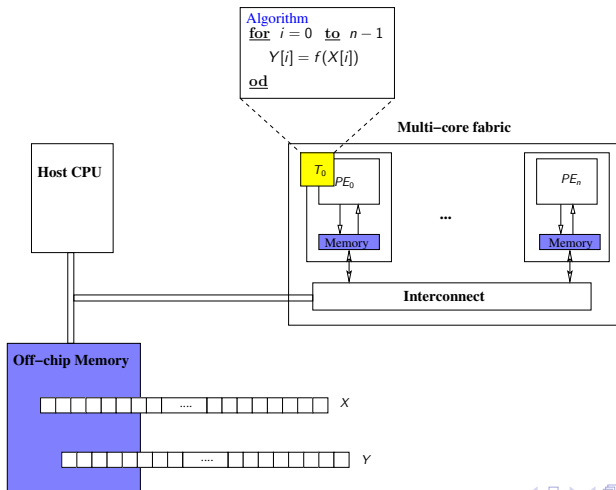
Heterogeneous Multi-core Architectures

- a powerful host processor and a multi-core fabric to **accelerate** computationally heavy kernels.



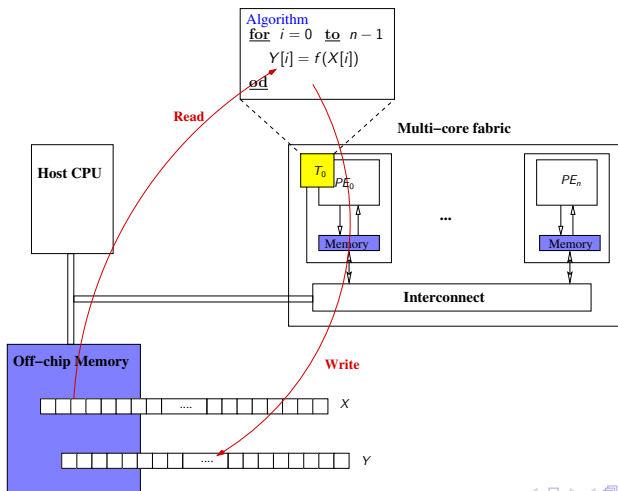
Data Transfers

- Offloadable kernels work on **large data sets**, initially stored in the **off-chip memory**.



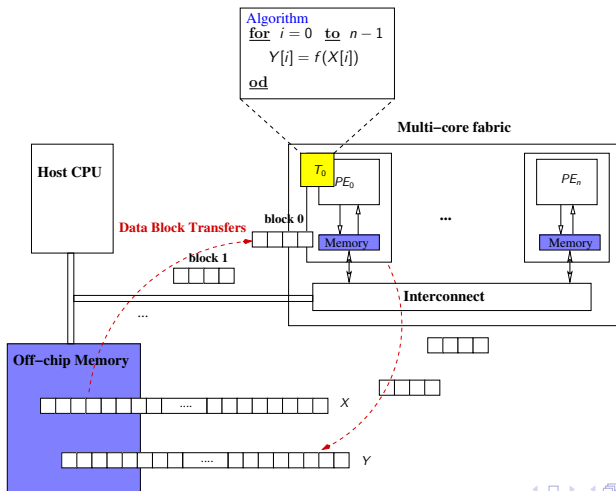
Data Transfers

- High off-chip memory latency: accessing off-chip data is **very costly**



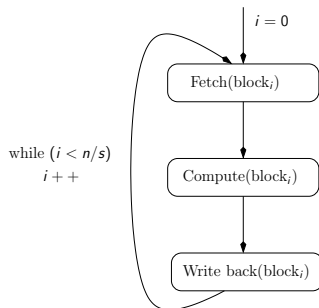
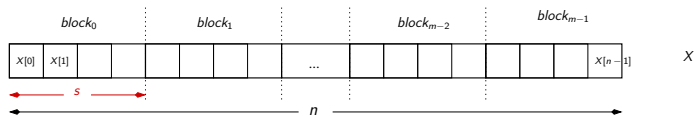
Data Transfers

- Data is transferred to a closer but smaller on-chip memory, using DMAs (Direct Memory Access).



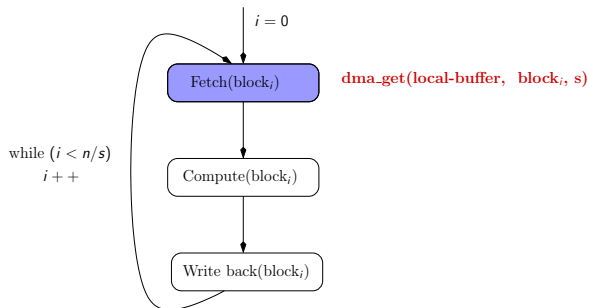
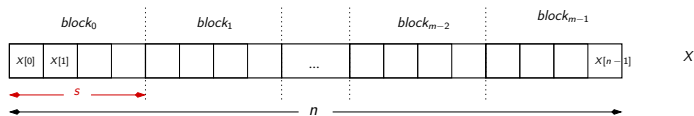
DMA Data Transfers

s : number of array elements in one block,



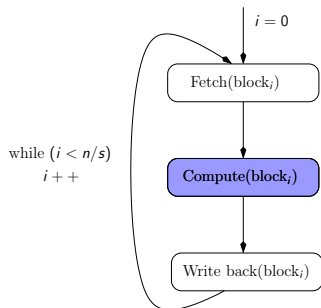
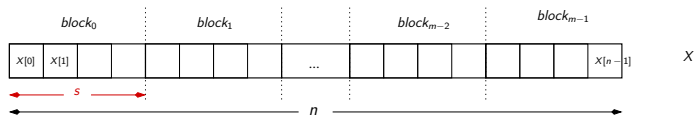
DMA Data Transfers

s : number of array elements in one block,



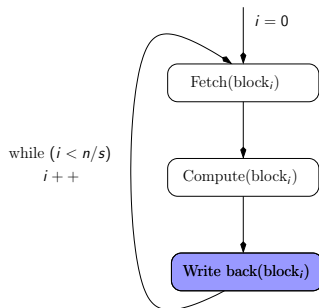
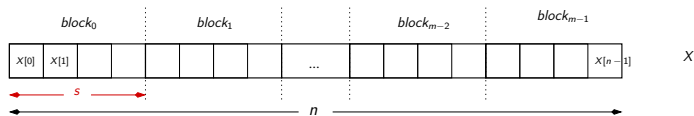
DMA Data Transfers

s : number of array elements in one block,



DMA Data Transfers

s : number of array elements in one block,

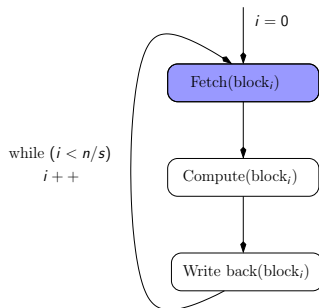
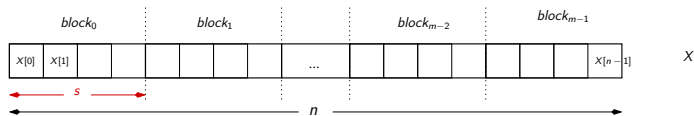


`dma_put(blocki, local-buffer, s)`



DMA Data Transfers

s : number of array elements in one block,



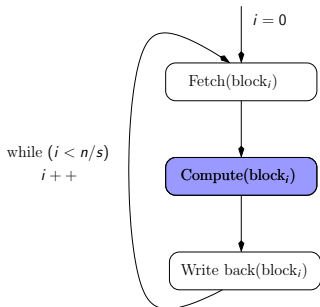
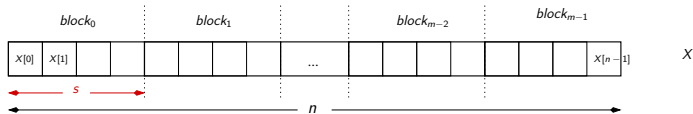
`dma_get(local-buffer, blocki, s)`

- Sequential execution of computations and data transfers.



DMA Data Transfers

s : number of array elements in one block,

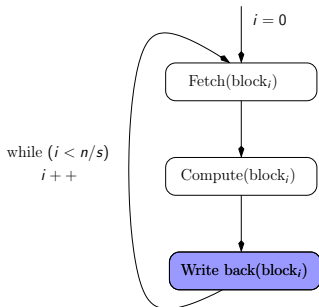
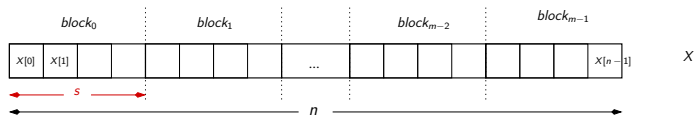


- Sequential execution of computations and data transfers.



DMA Data Transfers

s : number of array elements in one block,



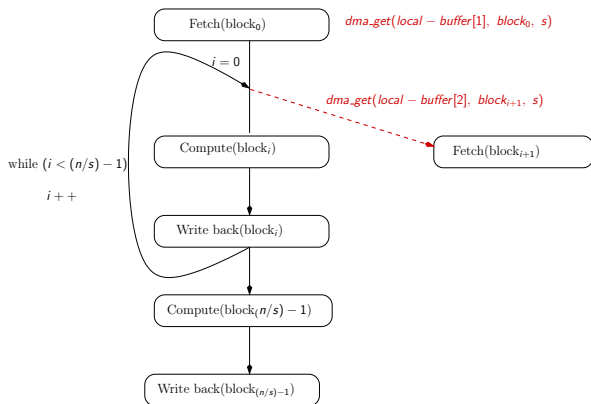
`dma_put(blocki, local-buffer, s)`

- Sequential execution of computations and data transfers.



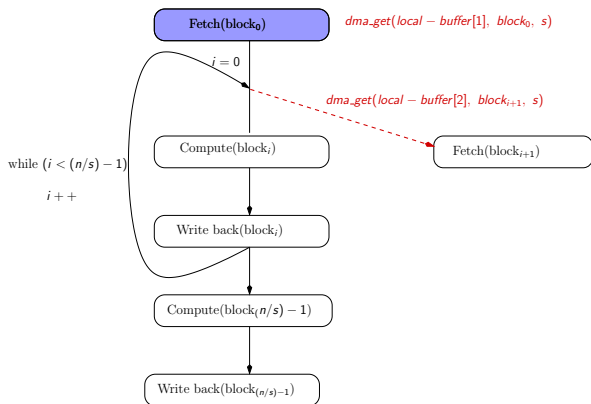
Double Buffering

Asynchronous DMA calls and double buffering:



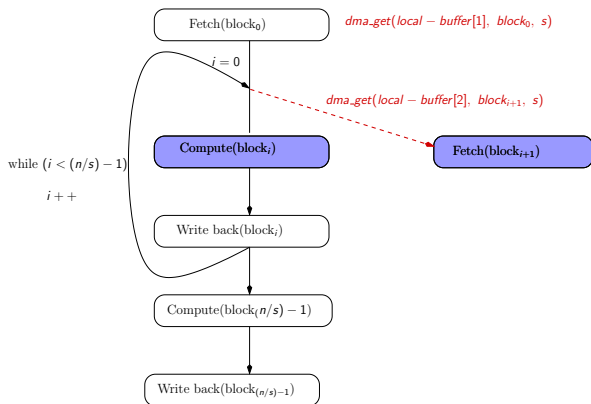
Double Buffering

Asynchronous DMA calls and double buffering:



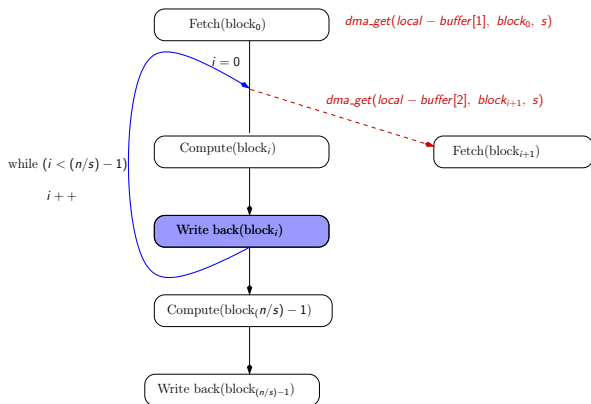
Double Buffering

Asynchronous DMA calls and double buffering:



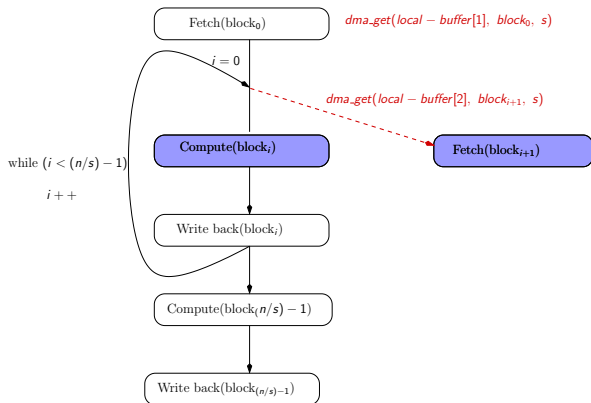
Double Buffering

Asynchronous DMA calls and double buffering:



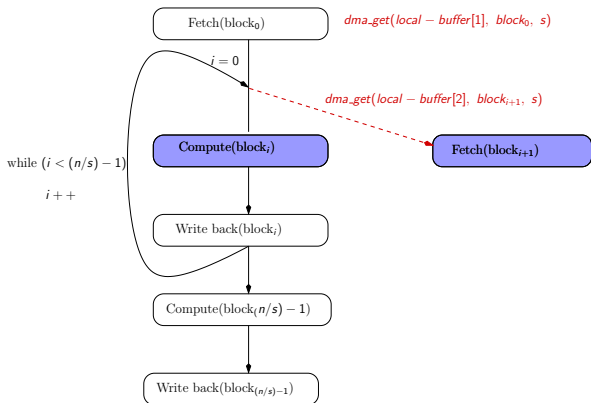
Double Buffering

Asynchronous DMA calls and double buffering:



Double Buffering

Asynchronous DMA calls and double buffering:

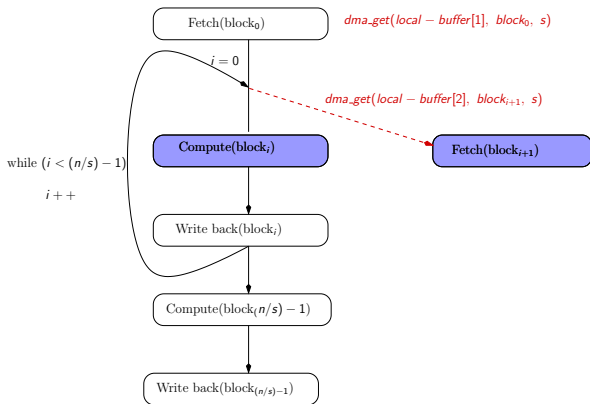


- Overlap of computations and data transfers.



Double Buffering

Asynchronous DMA calls and double buffering:



- Overlap of computations and data transfers.

What is the **choice s^*** that optimizes performance?



Our Contribution

- 1 We derive **optimal granularity** for DMA transfers given,
 - One processor.
 - Multiple processors.
- 2 We compare different strategies for **transferring shared data**



Outline

- 1 Introduction
- 2 Optimal Granularity
 - One Processor
 - Multiple Processors
- 3 Shared Data Transfers
- 4 Experiments on the CELL Architecture



Outline

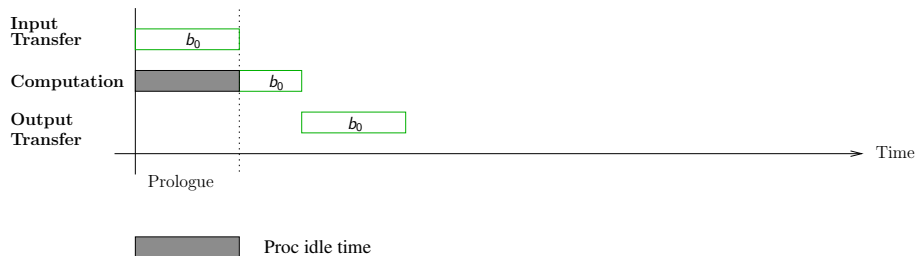
- 1 Introduction
- 2 Optimal Granularity
 - One Processor
 - Multiple Processors
- 3 Shared Data Transfers
- 4 Experiments on the CELL Architecture



Double Buffering Pipelined Execution

Overlap of,

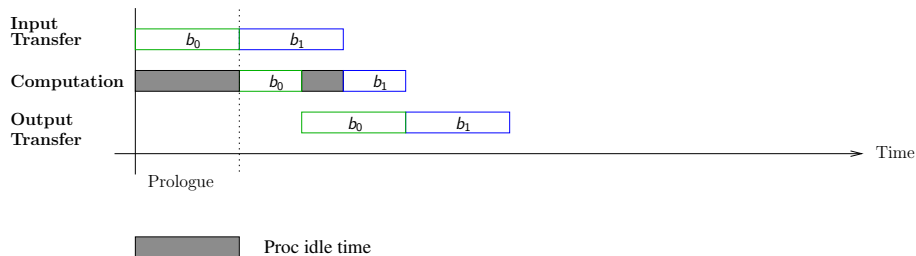
- *Computation* of **current block**,
- *Transfer* of **next block**.



Double Buffering Pipelined Execution

Overlap of,

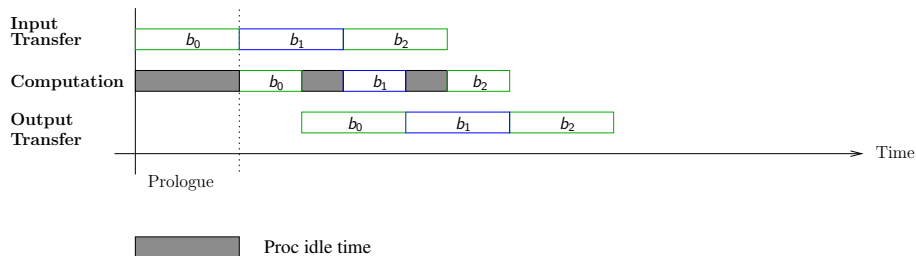
- *Computation* of **current block**,
- *Transfer* of **next block**.



Double Buffering Pipelined Execution

Overlap of,

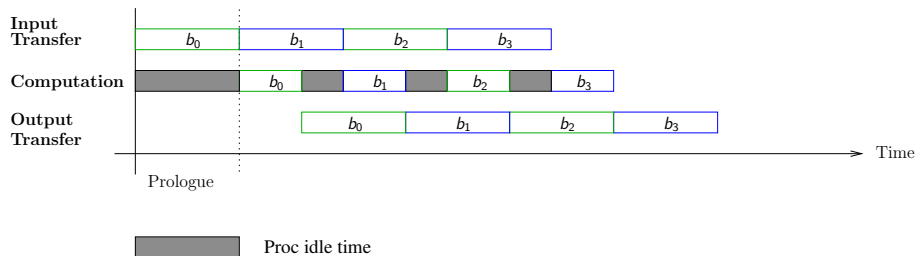
- *Computation* of **current block**,
- *Transfer* of **next block**.



Double Buffering Pipelined Execution

Overlap of,

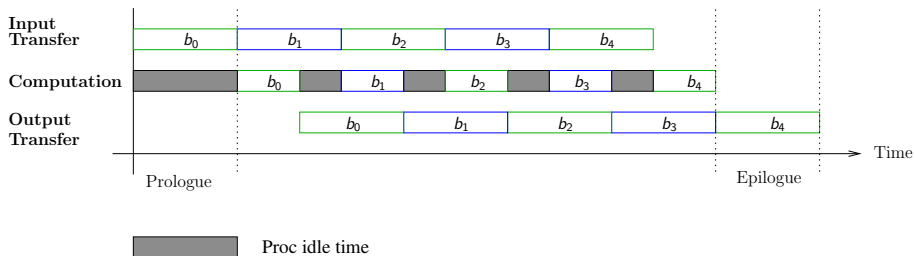
- *Computation* of **current block**,
- *Transfer* of **next block**.



Double Buffering Pipelined Execution

Overlap of,

- *Computation* of **current block**,
- *Transfer* of **next block**.



Computation and Transfer Time of a Data Block:

s : nb array elements clustered in one block,

Computation time $C(s)$:

DMA Transfer time $T(s)$:



Computation and Transfer Time of a Data Block:

s : nb array elements clustered in one block,

Computation time $C(s)$:

- ω : time to compute one element,

DMA Transfer time $T(s)$:



Computation and Transfer Time of a Data Block:

s : nb array elements clustered in one block,

Computation time $C(s)$:

- ω : time to compute one element,

$$C(s) = \omega \cdot s$$

DMA Transfer time $T(s)$:



Computation and Transfer Time of a Data Block:

s : nb array elements clustered in one block,

Computation time $C(s)$:

- ω : time to compute one element,

$$C(s) = \omega \cdot s$$

DMA Transfer time $T(s)$:

- I : fixed DMA initialization cost,



Computation and Transfer Time of a Data Block:

s : nb array elements clustered in one block,

Computation time $C(s)$:

- ω : time to compute one element,

$$C(s) = \omega \cdot s$$

DMA Transfer time $T(s)$:

- I : fixed DMA initialization cost,
- α : transfer cost per byte,



Computation and Transfer Time of a Data Block:

s : nb array elements clustered in one block,

Computation time $C(s)$:

- ω : time to compute one element,

$$C(s) = \omega \cdot s$$

DMA Transfer time $T(s)$:

- I : fixed DMA initialization cost,
- α : transfer cost per byte,
- b : size of one array element,



Computation and Transfer Time of a Data Block:

s : nb array elements clustered in one block,

Computation time $C(s)$:

- ω : time to compute one element,

$$C(s) = \omega \cdot s$$

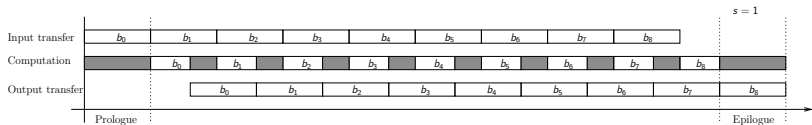
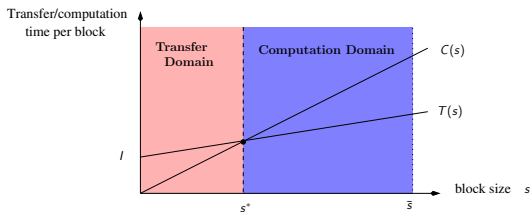
DMA Transfer time $T(s)$:

- l : fixed DMA initialization cost,
- α : transfer cost per byte,
- b : size of one array element,

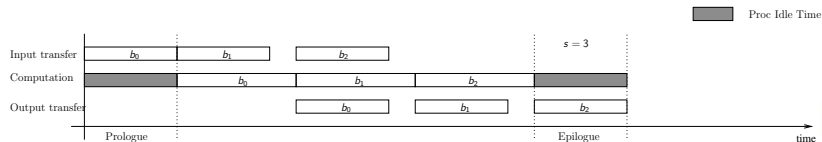
$$T(s) = l + \alpha \cdot b \cdot s$$



Computation and Transfer Ratio:



(a) Transfer Regime



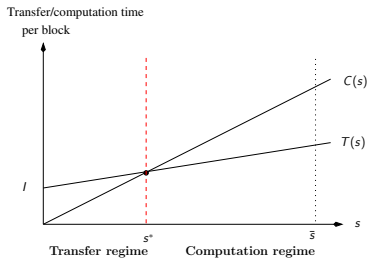
(b) Computation Regime



Optimal Granularity

Optimal Granularity s^* :

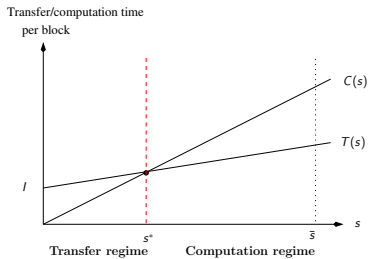
$$C(s^*) = T(s^*)$$



Optimal Granularity

Optimal Granularity s^* :

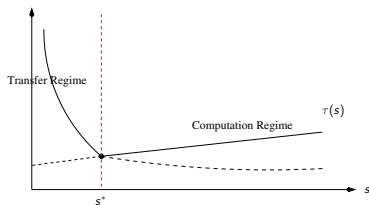
$$C(s^*) = T(s^*)$$



Pipeline execution time $\tau(s)$:

$$\tau(s) = \begin{cases} (n/s + 1) T(s) & \text{Transfer} \\ 2 \cdot T(s) + n \cdot \omega & \text{Computation} \end{cases}$$

Program execution time using double buffering



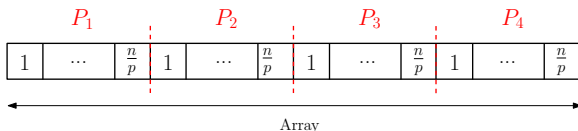
Outline

- 1 Introduction
- 2 Optimal Granularity
 - One Processor
 - Multiple Processors
- 3 Shared Data Transfers
- 4 Experiments on the CELL Architecture



Multiple Processors

- Partitioning: p contiguous chunks of data

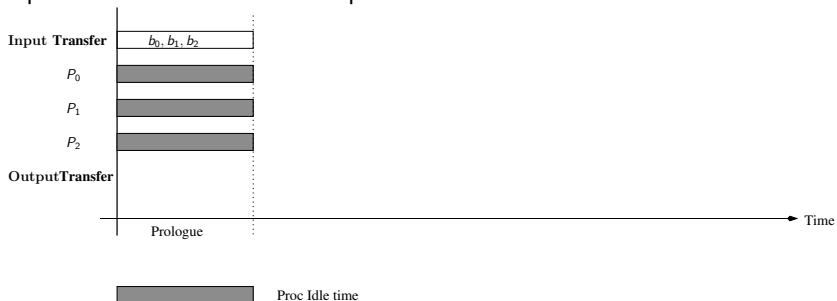


- Assumptions:
 - Processors are identical: same local store capacity, same double buffering granularity...etc.
 - A separate DMA per processor.



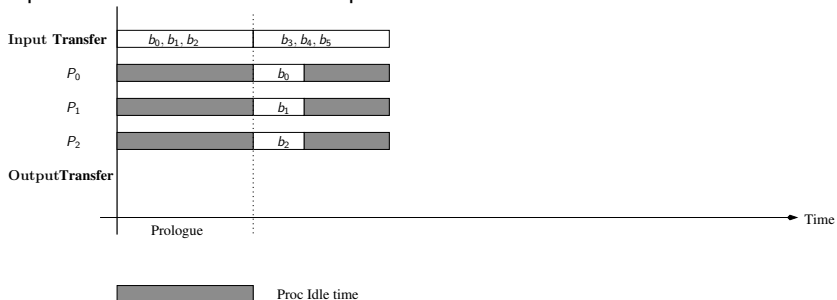
Multiple Processors

Pipelined execution for several processors:



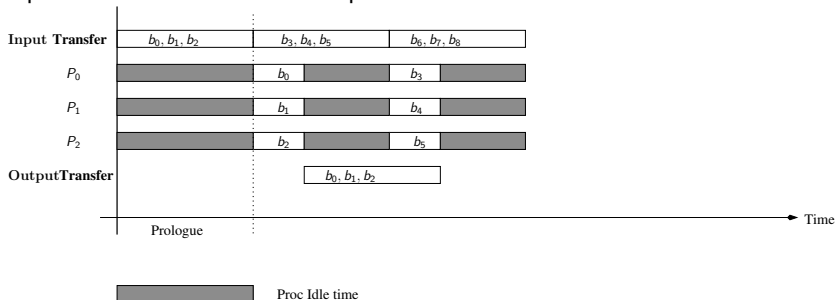
Multiple Processors

Pipelined execution for several processors:



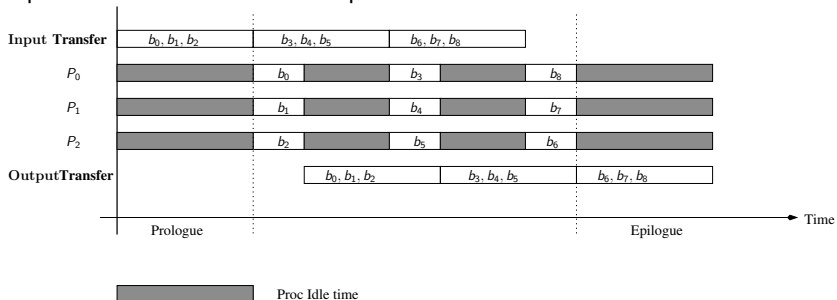
Multiple Processors

Pipelined execution for several processors:



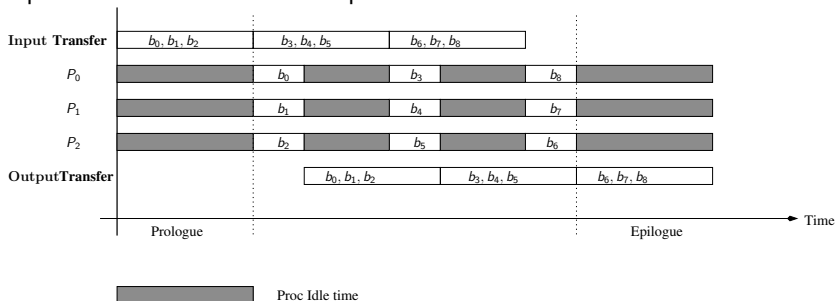
Multiple Processors

Pipelined execution for several processors:



Multiple Processors

Pipelined execution for several processors:

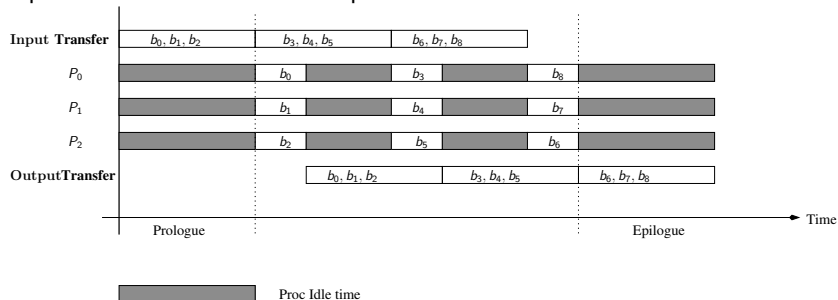


- Computation time of one block: $C(s) = \omega \cdot s$



Multiple Processors

Pipelined execution for several processors:



- Computation time of one block: $C(s) = \omega \cdot s$
- DMA transfer time of one block, given p processors,

$$T(s, p) = l + \alpha(p) \cdot b \cdot s$$

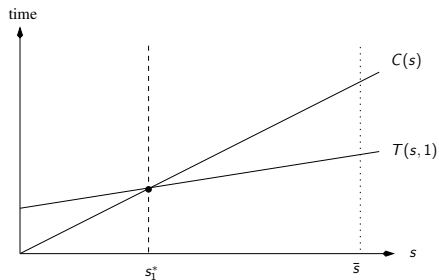
$\alpha(p)$: transfer cost per byte given **contentions** of p concurrent transfer requests.



Multiple Processors: Optimal Granularity

- Optimal granularity s_p^* ,

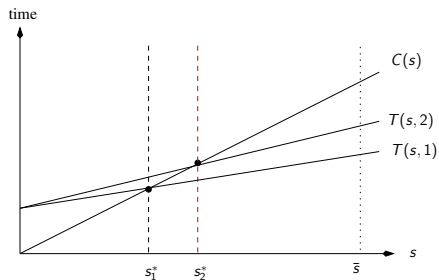
$$T(s_p^*, p) = C(s_p^*)$$



Multiple Processors: Optimal Granularity

- Optimal granularity s_p^* ,

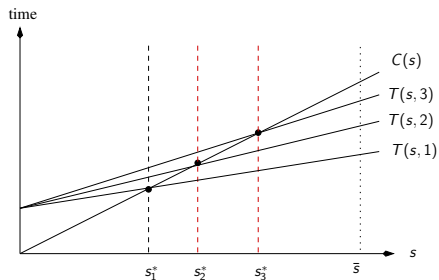
$$T(s_p^*, p) = C(s_p^*)$$



Multiple Processors: Optimal Granularity

- Optimal granularity s_p^* ,

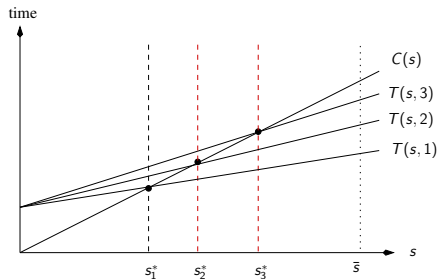
$$T(s_p^*, p) = C(s_p^*)$$



Multiple Processors: Optimal Granularity

- Optimal granularity s_p^* ,

$$T(s_p^*, p) = C(s_p^*)$$



Optimal Granularity **increases** with number of processors

Outline

- 1 Introduction
- 2 Optimal Granularity
 - One Processor
 - Multiple Processors
- 3 Shared Data Transfers
- 4 Experiments on the CELL Architecture



Applications with shared data

- Data parallel loop with **shared input data**:

```
for  $i := 0$  to  $n - 1$  do  
     $Y[i] := f(X[i], V[i]);$      $V[i] = \{X[i - 1], X[i - 2], \dots, X[i - k]\}$   
od
```



Applications with shared data

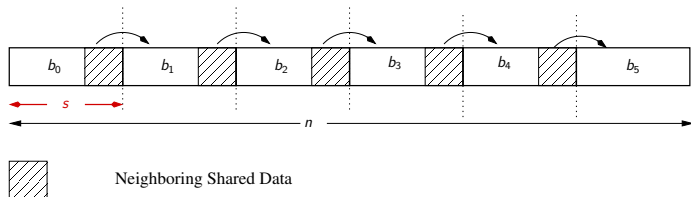
- Data parallel loop with **shared input data**:

for $i := 0$ **to** $n - 1$ **do**

$Y[i] := f(X[i], V[i]); \quad V[i] = \{X[i - 1], X[i - 2], \dots, X[i - k]\}$

od

- Neighboring blocks share data:



Strategies for transferring shared data

- 1 Replication
- 2 Inter-processor communication
- 3 Local buffering

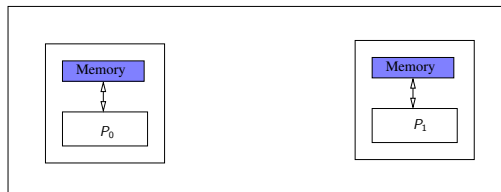
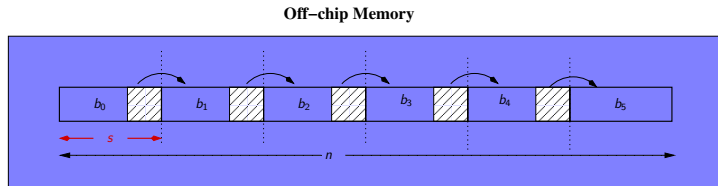


Strategies for transferring shared data

- 1 Replication
- 2 Inter-processor communication
- 3 Local buffering



1. Replication

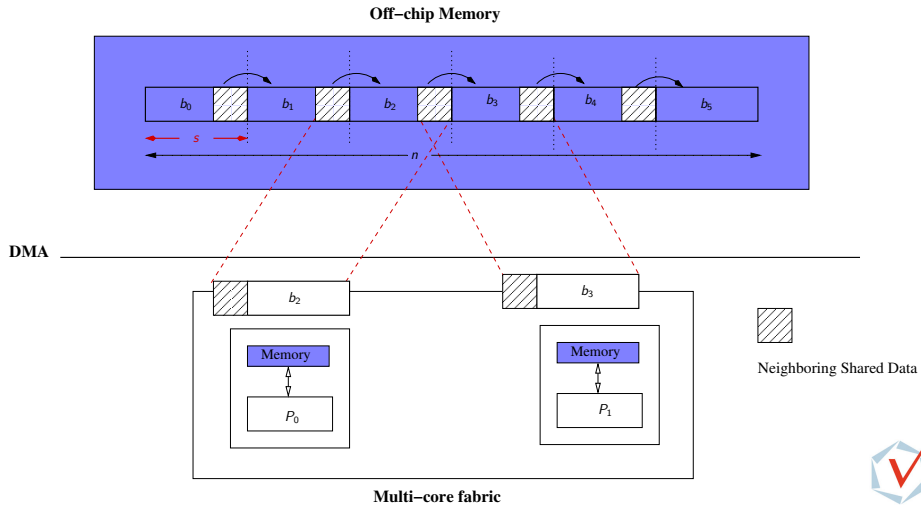


Neighboring Shared Data

Multi-core fabric



1. Replication



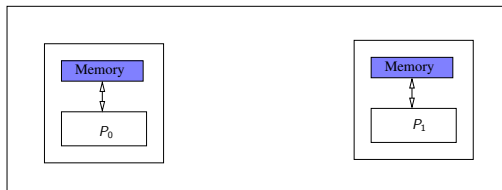
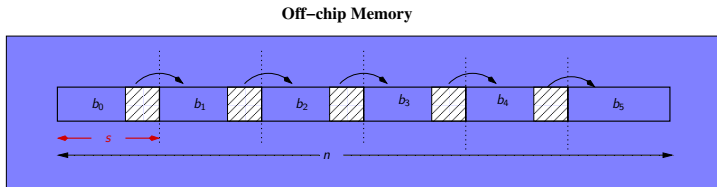
Strategies for transferring shared data

- 1 Replication
- 2 Inter-processor communication
- 3 Local buffering



2. Inter-processor communication:

Neighboring blocks computed neighboring processors,



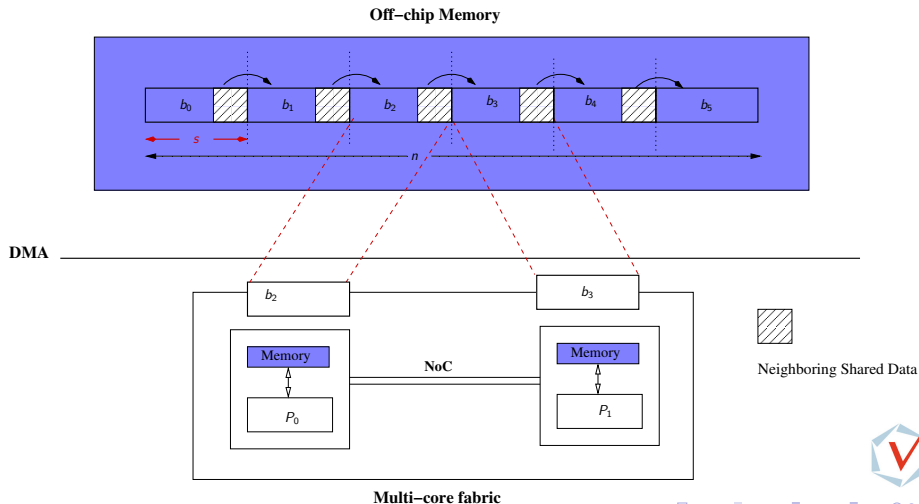
Neighboring Shared Data

Multi-core fabric



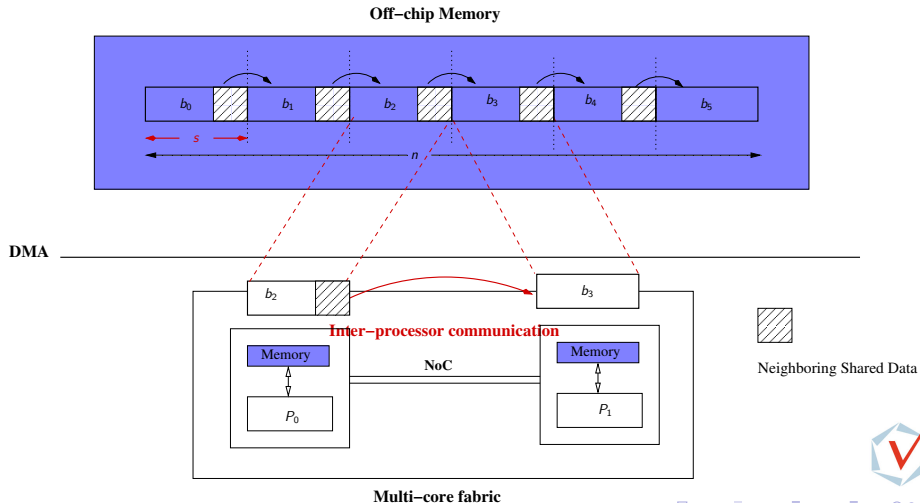
2. Inter-processor communication:

Neighboring blocks computed neighboring processors,



2. Inter-processor communication:

Neighboring blocks computed neighboring processors,



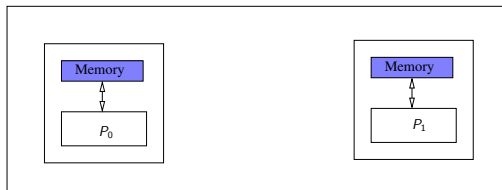
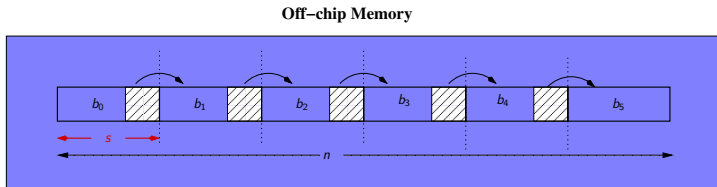
Strategies for transferring shared data

- 1 Replication
- 2 Inter-processor communication
- 3 Local buffering



3. Local Buffering

Neighboring blocks computed by the **same** processor,



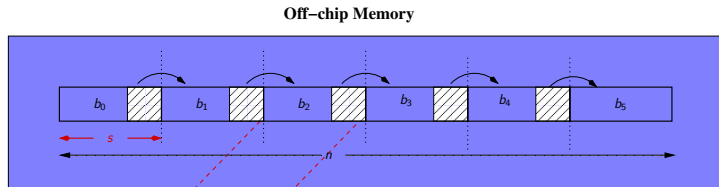
Neighboring Shared Data

Multi-core fabric

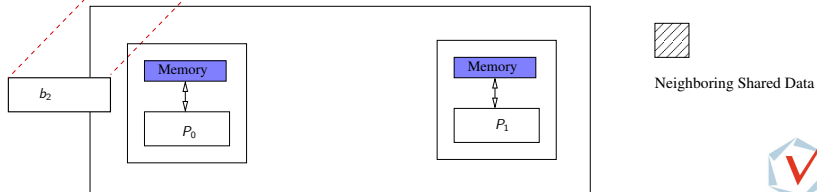


3. Local Buffering

Neighboring blocks computed by the **same** processor,



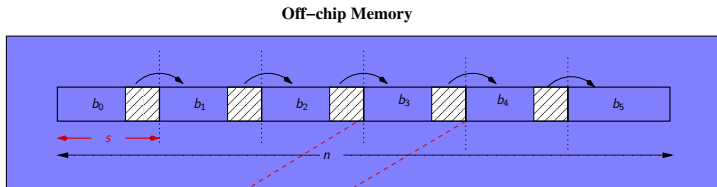
DMA



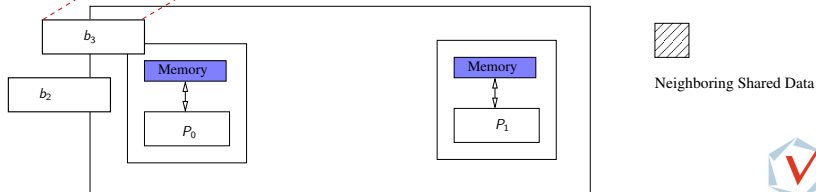
Multi-core fabric

3. Local Buffering

Neighboring blocks computed by the **same** processor,



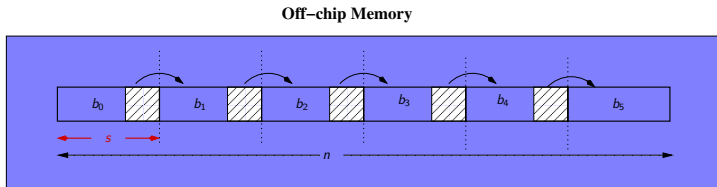
DMA



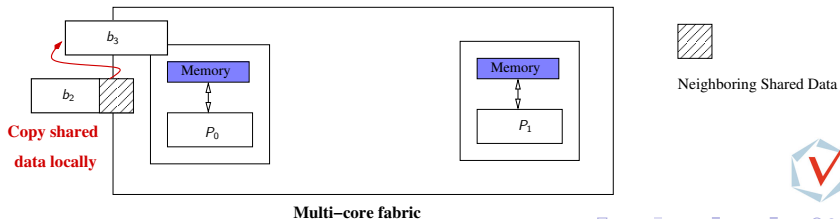
Multi-core fabric

3. Local Buffering

Neighboring blocks computed by the **same** processor,



DMA



Comparing Strategies

For each strategy,

- 1 we characterize the cost of transferring shared data,



Comparing Strategies

For each strategy,

- 1 we characterize the cost of transferring shared data,
- 2 we derive optimal granularity,



Comparing Strategies

For each strategy,

- 1 we **characterize** the **cost of transferring shared data**,
- 2 we derive **optimal granularity**,
- 3 we evaluate **overall execution time** in computation and transfer regimes.



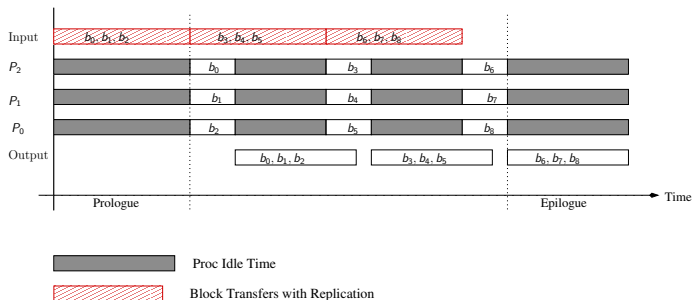
Comparing Strategies

Based on a **parametric** study, we derive **optimal strategy** for transferring shared data,



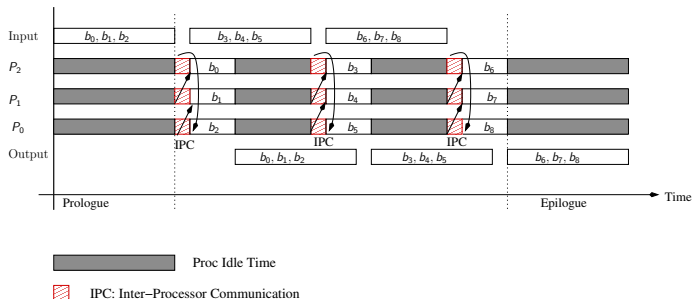
Comparing Strategies

- Replication:
 - ⊖ processors contentions overhead



Comparing Strategies

- Local Buffering and inter-processor communications:
 - ⊖ processing overhead

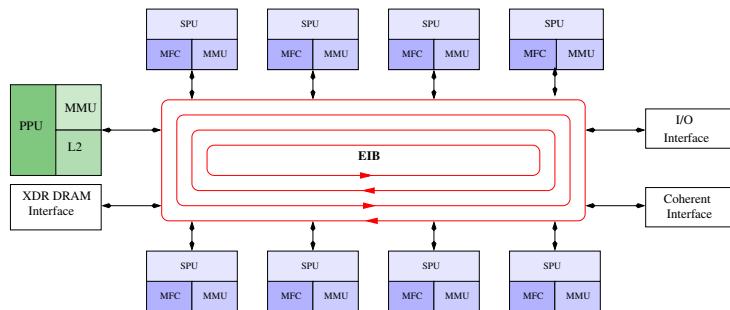


Outline

- 1 Introduction
- 2 Optimal Granularity
 - One Processor
 - Multiple Processors
- 3 Shared Data Transfers
- 4 Experiments on the CELL Architecture



Overview of Cell B.E. Architecture

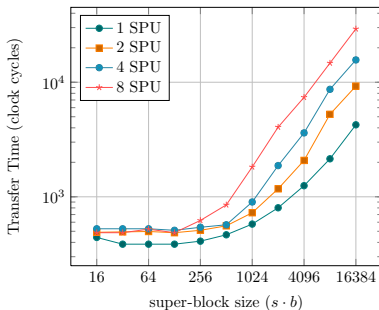


Platform Characteristics:

- 9-core heterogeneous multi-core architecture, with a Power Processor Element (PPE) and 8 Synergistic Processing Elements (SPE).
- Limited local store capacity per SPE: 256 Kbytes
- Explicitly managed memory system, using DMAs



Measured DMA Latency

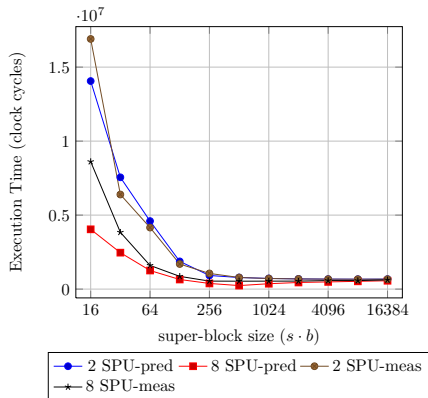


- Profiled hardware parameters:

DMA issue time	l	$\simeq 400$ clock cycles
Off-chip memory transfer cost/byte: 1 proc	$\alpha(1)$	0.22 clock cycles
Off-chip memory transfer cost/byte: p procs	$\alpha(p)$	$\simeq p \cdot \alpha(1)$
inter-processor comm transfer cost/byte for	β	0.13 clock cycles



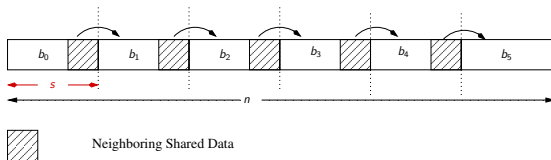
Optimal Granularity



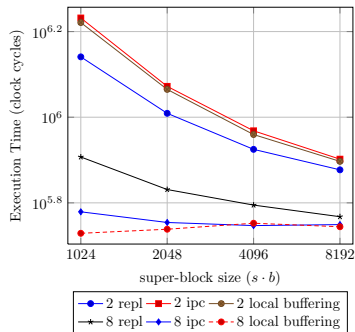
- predicted optimal granularities give good performance.



Shared Data Transfers



shared data size is 1024 bytes.



Conclusion

- We presented a general methodology for automating decisions about,
 - ① **Optimal granularity** for data transfers.
 - ② **Optimal strategy** for transferring shared data.
- We validated the experiments on the Cell architecture.
- On-going Work and Perspectives:,
 - ① Extend the work to other platforms, like P2012,
 - ② Extend the work to multidimensional data,
 - ③ Consider computation variabilities.

