Trace-based Performance Analysis Framework for Heterogeneous Multicore Systems

Shih-Hao Hung, *Chia-Heng Tu*, Thean-Siew Soon

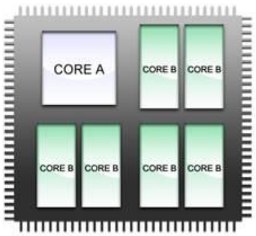
Performance, Applications and Security (PAS) Lab
CSIE & GINM, National Taiwan University

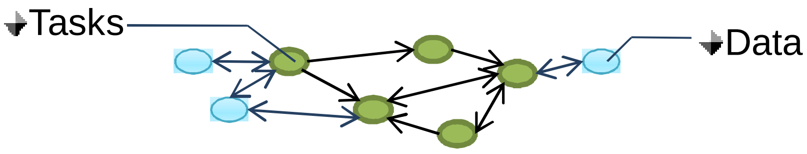
Outlines

- Motivation
- Designs of the ParallelTracer
- Experimental Results and Evaluation
- Summary

Motivation (1/2)

- The complexity of Today's computer systems have increased with the advent of embedded heterogeneous multicore systems
- A good application developer has to be familiar with the heterogeneity
  - Architecture
    - Multiple Instruction Set Architectures (ISAs)
    - Communication schemes on the platform
  - Application
    - Partitioning of computation and load balance
    - Communication patterns of the application





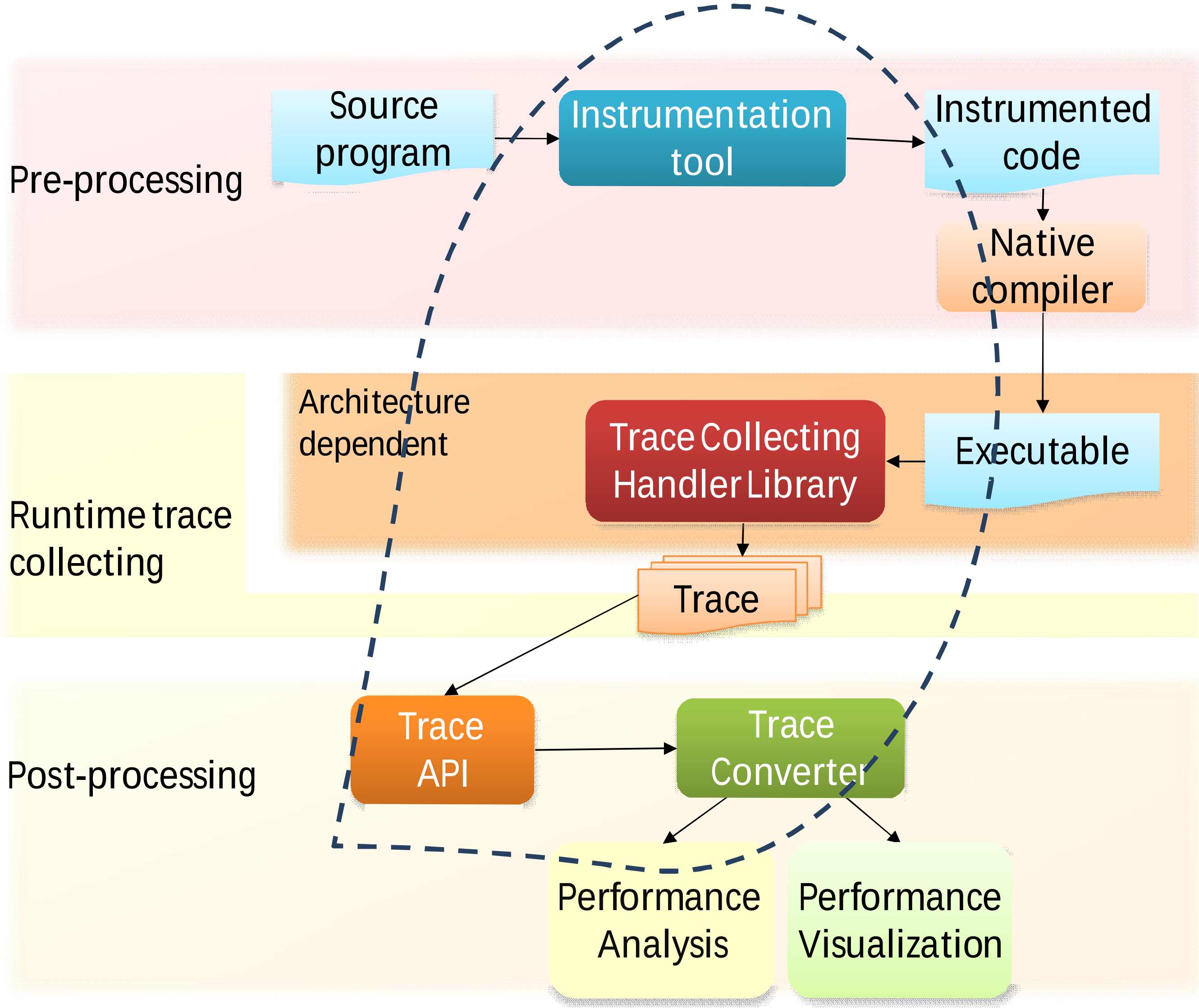
Motivation (2/2)

- Performance analysis tools are essential in exploring the multicore architecture's potential
  - Tracing tools can reveal detailed machine-application interactions
  - Most tracing tools are designed for homogeneous multicore systems and platform-dependent
- Our goals:
  - Develop a portable toolkit for embedded heterogeneous multicore platform
  - Ported our toolkit to a heterogeneous multicore system, IBM Cell, as a case study to demonstrate the efficiency of our toolkit, *ParallelTracer*

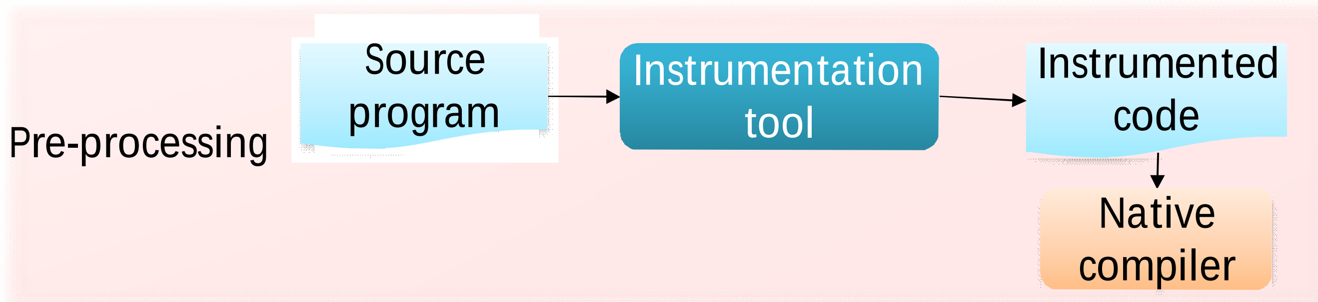
Outlines

- Motivation
- Designs of the ParallelTracer
- Experimental Results and Evaluation
- Summary

ParallelTracer Overview



Pre-Processing



- Injects probes at the source level
  - Capable of tracing events specified by the user
  - By default, it is pre-programmed to identify:
    - Program flow (function entry and exit)
    - Communication events (DMA, send, recv, put, get, …)
- Heavy instrumentation can interfere with normal program execution!
  - Need to minimize trace collection overhead

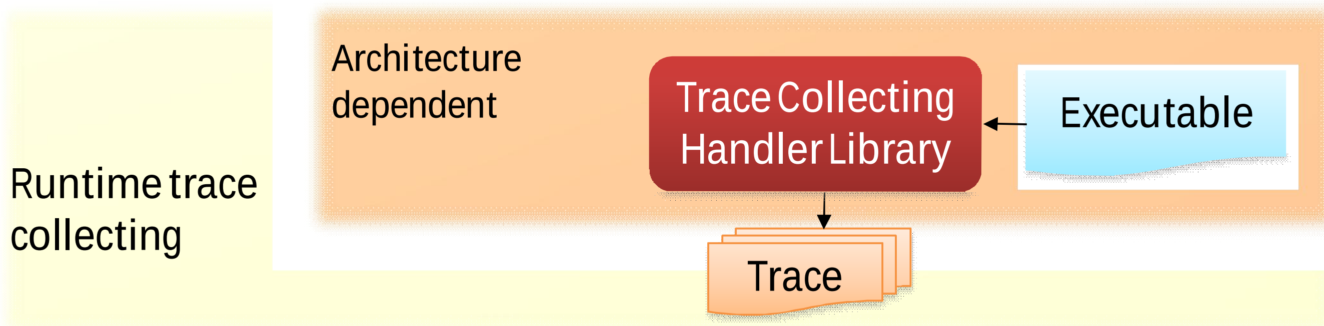
Screenshots of Original Source Code vs. Instrumented Code

```
int main (unsigned long long speid,unsigned long long
argp,unsigned long long envp)
{

    ............
  /* Get job from Master thread (PPE) */
  mfc_get(&mystruct,(unsigned int) argp,128,tag,0,0);

   /* Wait until  data  received */
  mfc_read_tag_status_all();

   /* Do the computation */
    ...............

  /*  Return data to Master thread (PPE) */
  mfc_put(Z,(unsigned long int) mystruct.Z+offset,128,tag,0,0);

  /* Wait until  all data has been sent */
  mfc_read_tag_status_all();



  return 0;
} // Example code at Data Processor side (SPE)
```

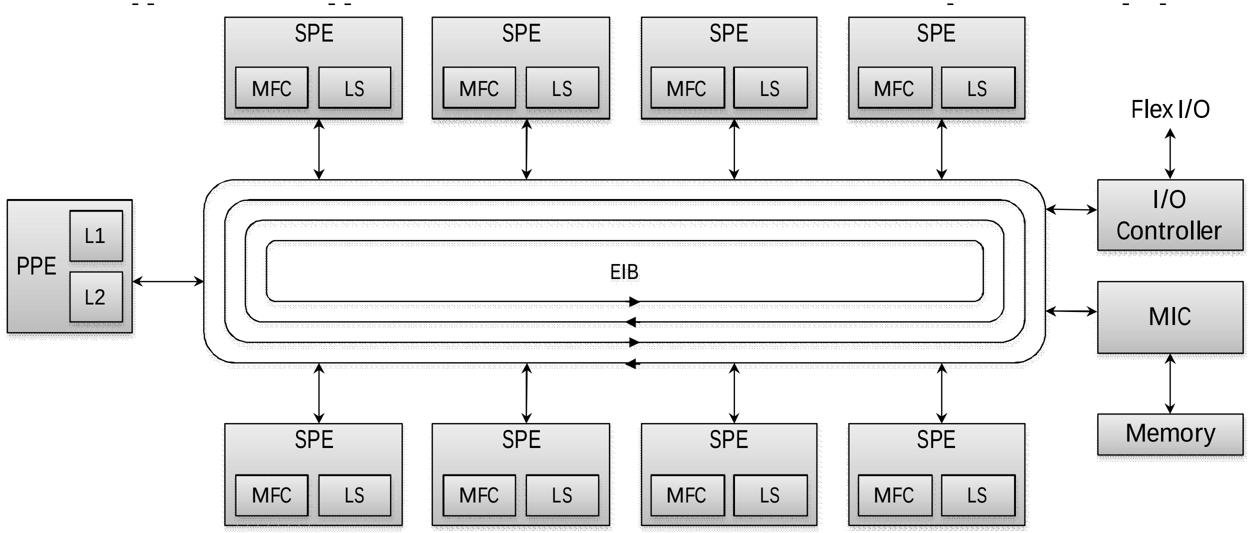
```
int main (unsigned long long speid,unsigned long long
argp,unsigned long long envp)
{
   SPUTraceInit();
    ............
   /* Get job from Master thread (PPE) */
   mfc_get(&mystruct,(unsigned int) argp,128,tag,0,0);
   DMAStart(tag, DMA_GET, 128);
   /* Wait until data received */
  mfc_read_tag_status_all();
   DMAEnd();

    /* Do the computation */
    ...............
  /*  Return data to Master thread (PPE) */
  mfc_put(Z,(unsigned long int) mystruct.Z+offset,128,tag,0,0);
  DMAStart(tag, DMA_PUT, 28);
  /* Wait until  all data has been sent */
  mfc_read_tag_status_all();
  DMAEnd();

  SPUTraceTerminate();
  return 0;
}
```

The monitoring codes are inserted into the source code right before/after the interested communication functions, i.e., DMA

Runtime Trace Collection

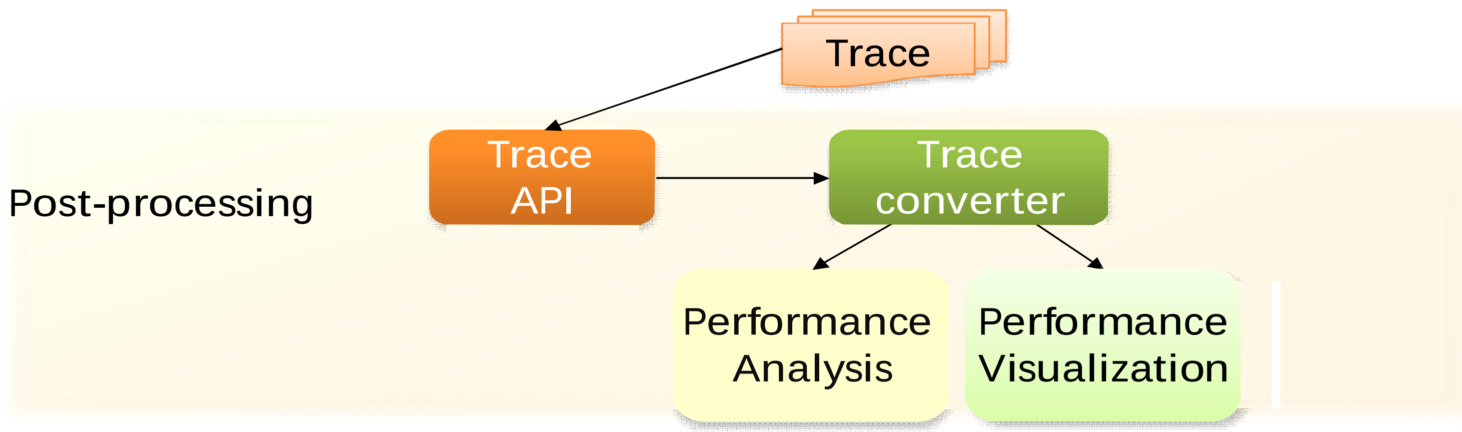


- Trace collection
  -
  -
  - ed to



- O

Post-Processing



- Trace API
  - Unified interface to access trace data
- Trace converter
  - Converting the trace data format
- Visualization tool
  - Communication graph
  - Timeline diagram

Outlines

- Motivation
- Designs of the ParallelTracer
- Experimental Results and Evaluation
- Summary

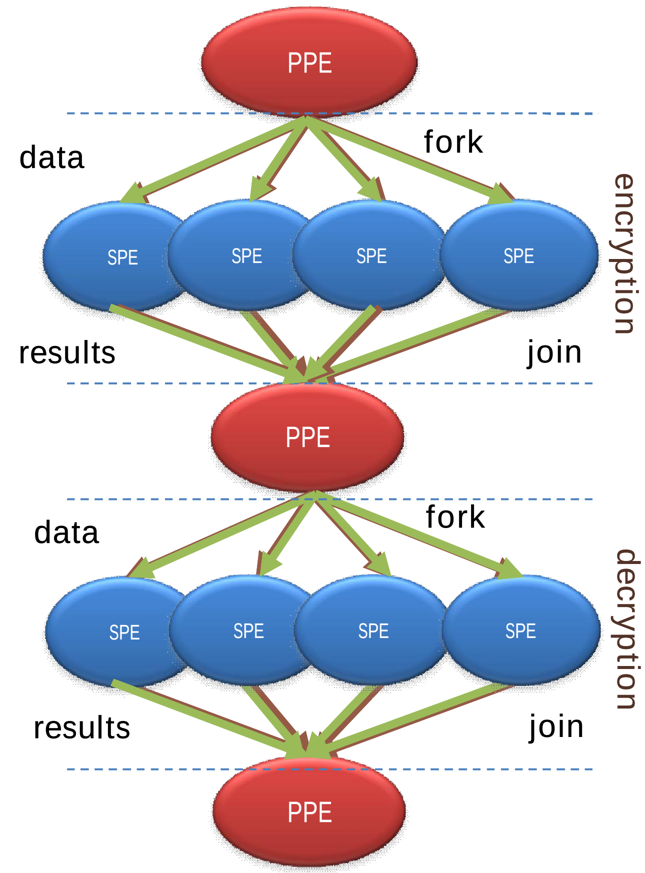
Experimental Setup

- IBM BladeCenter QS21*
  - Cell processor chip x 2
    - Clock rate: 3.2GHz
    - Number of cores: 2 PPE + 16 SPEs
    - Main memory: 2GB
  - RedHat Enterprise Linux 5.1
    - Linux Kernel 2.6.18
    - GCC compiler 4.1.1
    - Cell SDK 3.0

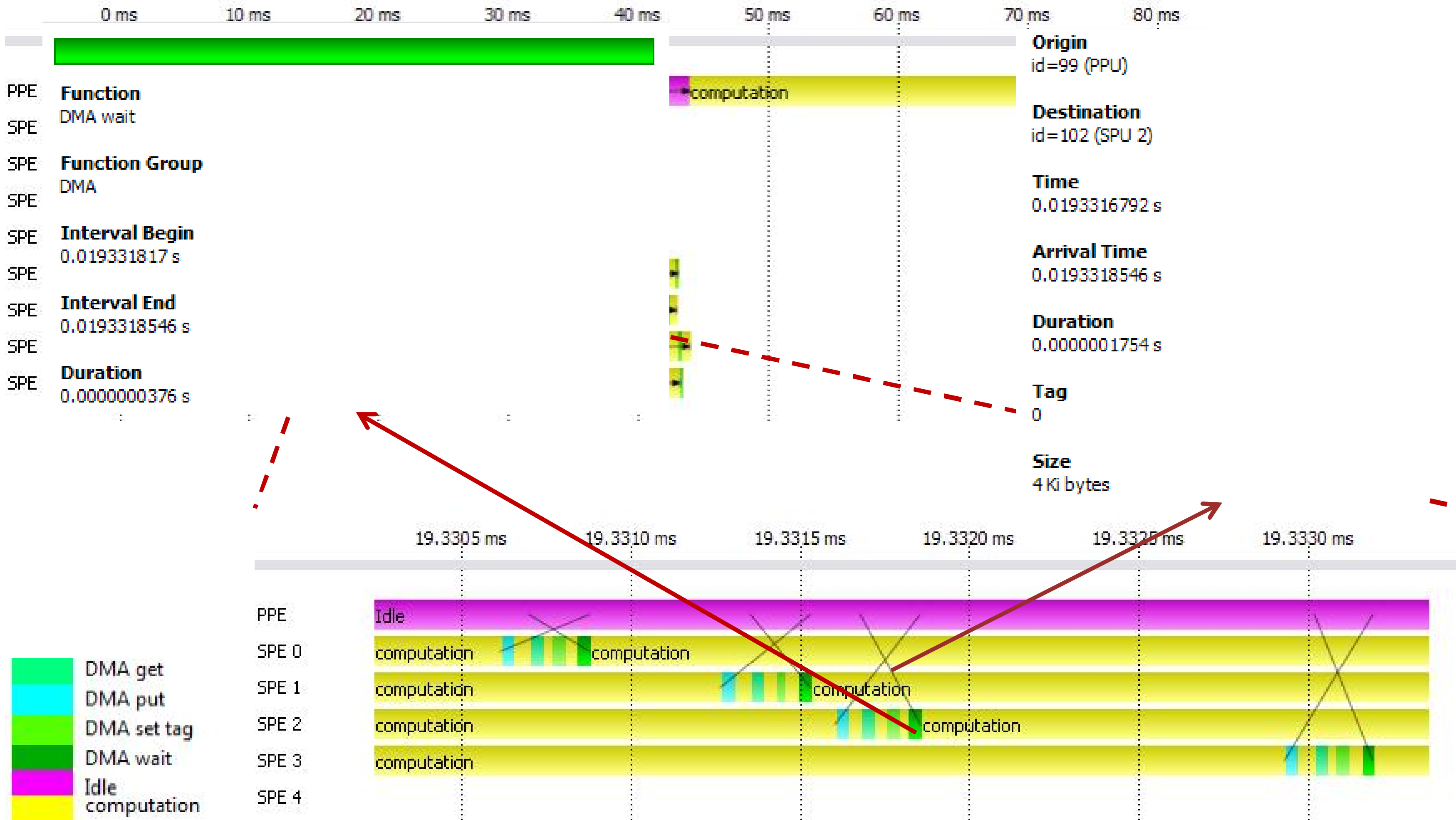
*IBM bladecenter qs21. http://www-03.ibm.com/systems/bladecenter/hardware/servers/qs21/

Case Study: A Data Parallel Application

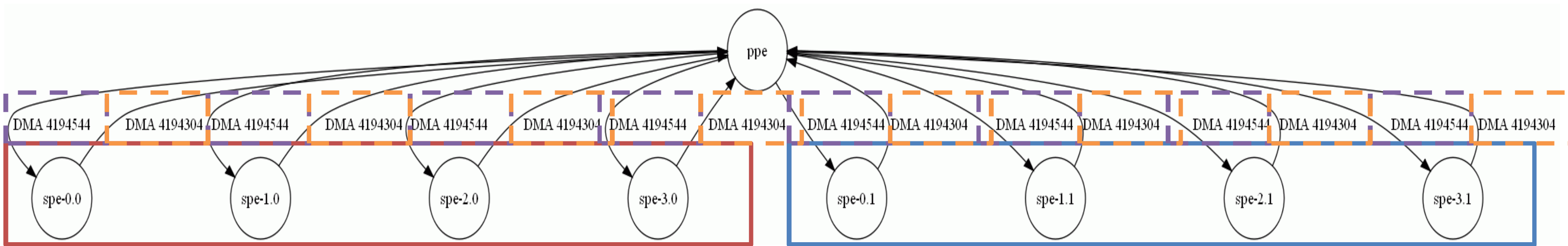
- RC5 (block cipher) is used to capture the behaviors of a data parallel application
  - 1 master thread (PPE)
  - 4 worker threads (SPEs)
  - Data size
    - 4MB of *int* type data (16MB)



RC5 – Timeline Diagram

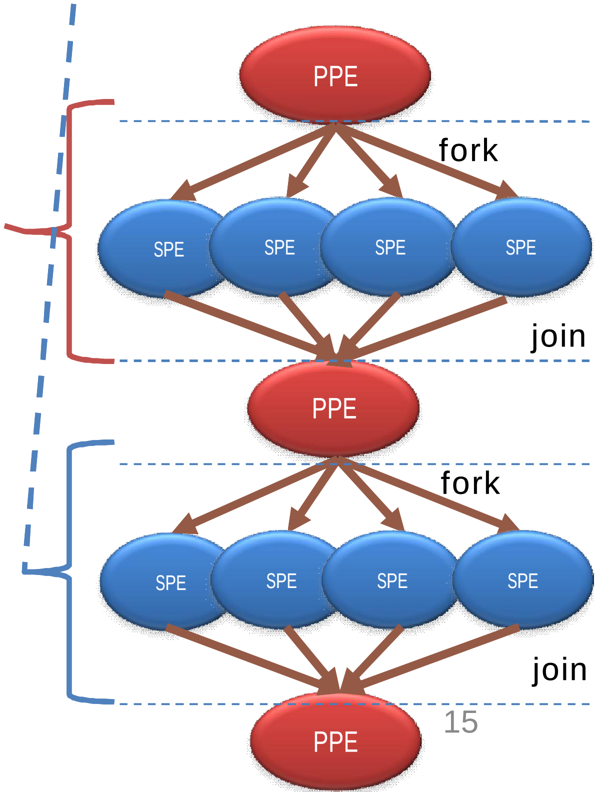


RC5 – Communication Graph



- Capturing communication patterns
  - Source and destination
  - Communication schemes (e.g., DMA)

- Indicates DMA PUT operations
- Each SPE gets 4MB data from PPE throughout the application execution



Performance Overhead

- Overhead of trace collection
  - Without optimization: 37.71%
  - Double buffering and asynchronous I/O: 10.01%

Outlines

- Motivation
- Designs of the ParallelTracer
- Experimental Results and Evaluation
- Summary

Summary

- We developed a portable tracing toolkit for embedded heterogeneous multicore platforms
  - Support tracing capabilities of recording:
  - Communication and computation events
  - User specific events

- Our experiment results on Cell show that our tracing tool generates low overhead
  - Average ~10% performance overhead

THANKS FOR YOUR ATTENTION!