2008-01-17

DS-CDMA Procedures
with the Cell Broadband Engine
Project Exam - Group 07gr942

Jes Toft Kristensen
Peter August Simonsen

9th SEMESTER
APPLIED SIGNAL PROCESSING AND IMPLEMENTATION

January 11th, 2008

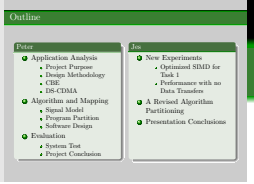
DS-CDMA Procedures with the Cell Broadband Engine

Project Exam - Group 07gr942

Jes Toft Kristensen
Peter August Simonsen

9^{th} SEMESTER
APPLIED SIGNAL PROCESSING AND IMPLEMENTATION

January 11^{th}, 2008



Present yourself

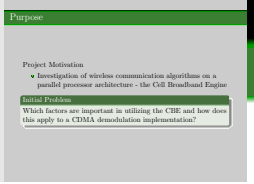
Outline

Peter

1. Application Analysis
   - Project Purpose
   - Design Methodology
   - CBE
   - DS-CDMA
2. Algorithm and Mapping
   - Signal Model
   - Program Partition
   - Software Design
3. Evaluation
   - System Test
   - Project Conclusion

Jes

4. New Experiments
   - Optimized SIMD for Task 1
   - Performance with no Data Transfers
5. A Revised Algorithm Partitioning
6. Presentation Conclusions



- Project proposed by R & S to see if CBE could be attractive for wireless communications applications.
-

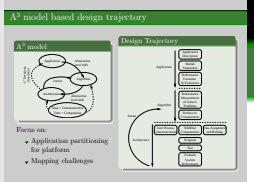
Purpose

Project Motivation

- Investigation of wireless communication algorithms on a parallel processor architecture - the Cell Broadband Engine

Initial Problem

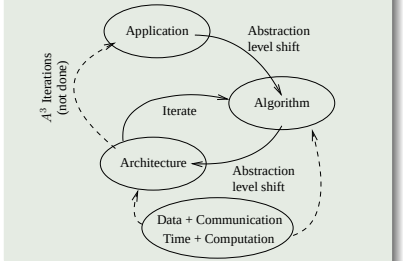
Which factors are important in utilizing the CBE and how does this apply to a CDMA demodulation implementation?



- Purpose:
  Structure focus areas
- walk through abstraction levels
  - Application: **description/simulation of the problem**
  - Algorithm: **general rules for alg. dev. for arch.** applied to problem
  - Architecture: **the actual implementation**

A^3 model based design trajectory

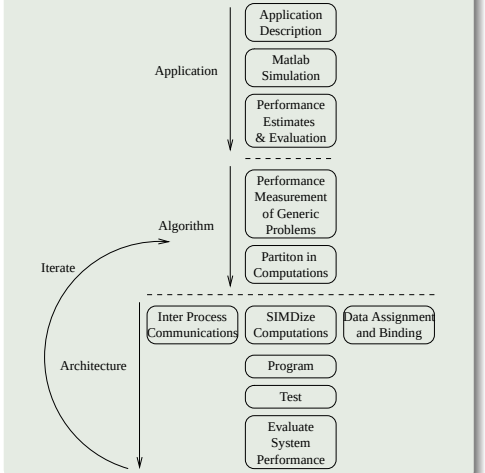
A^3 model

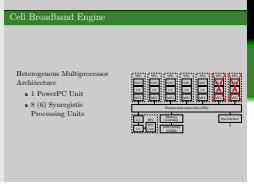


Focus on:

- Application partitioning for platform
- Mapping challenges

Design Trajectory



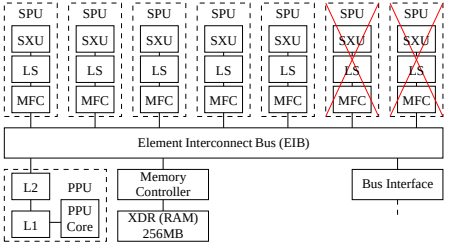


Cell Broadband Engine

- Standard 64 bit dual core general purpose PowerPC unit.
- Allows the CBE to function as an **ordinary PC**
- **8 data processors** connected to a PPC by a common element interconnect bus
- SPU consists of **SXU, LS, MFC**
- PS3 only has 6 available SPUs, one is defect, one is reserved for game OS

Heterogenous Multiprocessor Architecture

- 1 PowerPC Unit
- 8 (6) Synergistic Processing Units



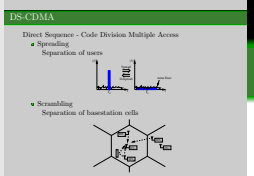
Mapping Challenges
Main issues in mapping
SIMD instruction utilization
Eliminate conditional branches
Concurrent data transfers and data processing

Mapping Challenges

- SIMD for **pipeline utilization**. Done by **vectorizing algorithm**
- No conditional branches. **NO control structures**
- **Utilization** of all funcitonal units

Main issues in mapping

- SIMD instruction utilization
- Eliminate conditional branches
- Concurrent data transfers and data processing

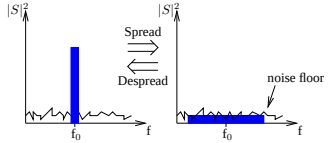


- walk through figure
- Spreading
  Used for **separation of users** within basestation cells by code assignment
  Each symbol multiplied by af sequence.
  Effectively spreads signal in frequency spectrum
- Scrambling
  Used for **separation of basestation cells** by whitening of spread signals
- explain that **spreading and scrambling codes are known sequences** can be multiplied first, (sequence of multiplications are not important)

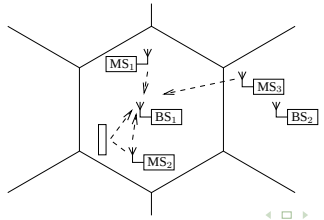
DS-CDMA

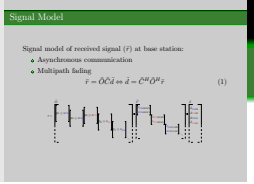
Direct Sequence - Code Division Multiple Access

- Spreading
  Separation of users



- Scrambling
  Separation of basestation cells





- Async comm results in **displacements in O**
- Multipath fading results in **multiple copies of pos in O and vectors in C**
- when **O is orthogonal** O **inverse equals transpose**.

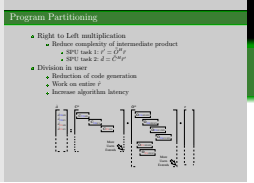
Signal Model

Signal model of received signal ($\bar{r}$) at base station:

- Asynchronous communication
- Multipath fading

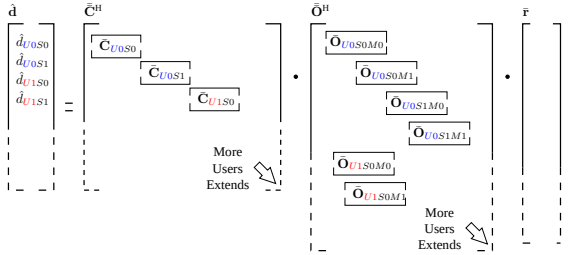
$$\bar{r} = \bar{\bar{O}}\bar{\bar{C}}\bar{d} \Leftrightarrow \hat{d} = \bar{\bar{C}}^H\bar{\bar{O}}^H\bar{r} \quad (1)$$

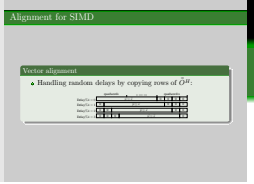
$$\bar{r} = \begin{bmatrix} \bar{O} \\ \begin{bmatrix} \bar{p}_0 \odot \bar{s}_{U0} \end{bmatrix} \begin{bmatrix} \bar{p}_0 \odot \bar{s}_{U0} \end{bmatrix} \begin{bmatrix} \bar{p}_0 \odot \bar{s}_{U1} \end{bmatrix} \begin{bmatrix} \bar{p}_0 \odot \bar{s}_{U1} \end{bmatrix} \begin{bmatrix} \bar{p}_1 \odot \bar{s}_{U0} \end{bmatrix} \ddots \end{bmatrix} \bullet \begin{bmatrix} \bar{C} \\ \begin{bmatrix} c_{U0S0M0} \\ c_{U0S0M1} \end{bmatrix} \begin{bmatrix} c_{U1S0M0} \\ c_{U1S0M1} \end{bmatrix} \begin{bmatrix} c_{U0S1M0} \\ c_{U0S1M1} \end{bmatrix} \ddots \end{bmatrix} \bullet \begin{bmatrix} \bar{d} \\ \begin{bmatrix} d_{U0S0} \\ d_{U1S0} \\ d_{U0S1} \\ d_{U1S1} \end{bmatrix} \end{bmatrix}$$



Program Partitioning

- Right to Left multiplication
  - Reduce complexity of intermediate product
    - SPU task 1: $\bar{r}' = \bar{\bar{O}}^H \bar{r}$
    - SPU task 2: $\hat{d} = \bar{\bar{C}}^H \bar{r}'$
- Division in user
  - Reduction of code generation
  - Work on entire $\hat{r}$
  - Increase algorithm latency



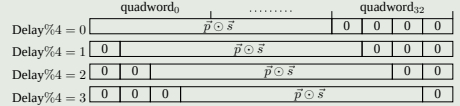


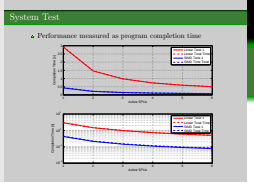
- **Init operations** is generation of pos based on assigned user number.
  4 copies for data **quadword alignment**

Alignment for SIMD

Vector alignment

- Handling random delays by copying rows of $\bar{\bar{O}}^H$:

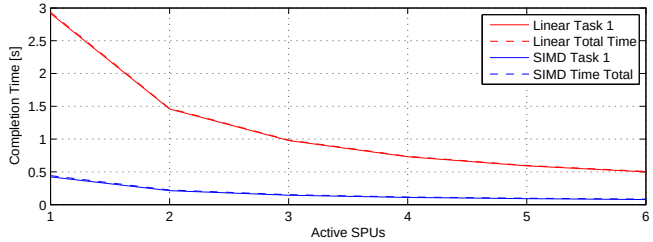


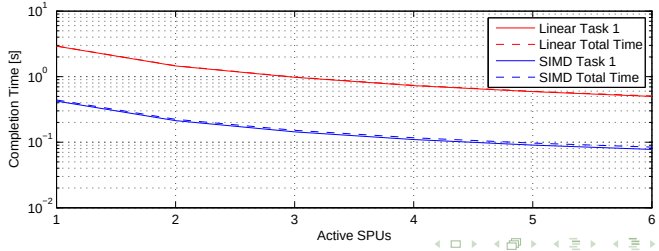


System Test

- Test run with **variable number of active SPUs**
- Done to determine applications ability to **Scale for parallel processing**
- Rerun (100x) for elimination of uncertainty
- System performance doesn't scale linearly with no. of active SPUs

- Performance measured as program completion time



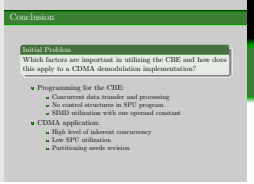


Results Discussion
- 10 ms completion requirement
  - 84.2 ms achieved
  - 10.3% SPU utilization
- Problems origin:
  - Inefficient pipeline utilization
  - Suboptimal compiler scheduling

- loading of **2 operands** for each mult is inefficient.
- compiler scheduling results in **40% pipeline stalls**, need for getting hands dirty and do it manually

Results Discussion

- 10 ms completion requirement
  - 84.2 ms achieved
  - 10.3% SPU utilization
- Problems origin:
  - Inefficient pipeline utilization
  - Suboptimal compiler scheduling



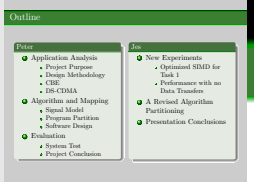
- Programming for the CBE:
  - **Concurrent** data transfer and processing
  - **No control structures** in SPU program
  - SIMD utilization with **one operand constant**
- CDMA application:
  - High level of **inherent concurrency**
  - Low SPU utilization
  - Partitioning needs revision

Conclusion

Initial Problem

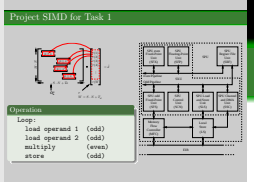
Which factors are important in utilizing the CBE and how does this apply to a CDMA demodulation implementation?

- Programming for the CBE:
  - Concurrent data transfer and processing
  - No control structures in SPU program
  - SIMD utilization with one operand constant
- CDMA application:
  - High level of inherent concurrency
  - Low SPU utilization
  - Partitioning needs revision



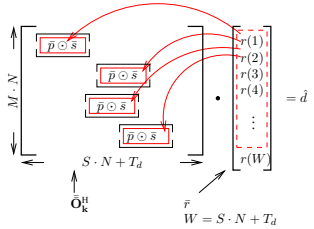
Present yourself

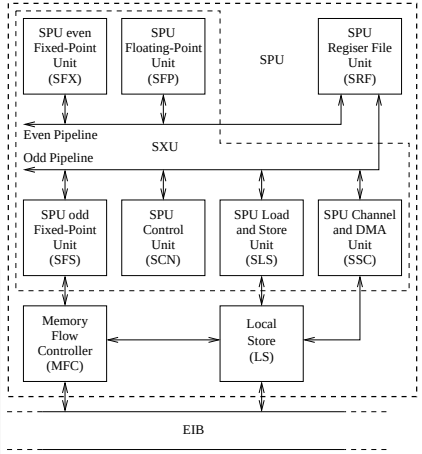
Outline



- **Describe architecture**
- **Suboptimal** Scheduling of Task 1 **(4 and 4, restart)**
- This is **interesting** as it would be an issue in further work with the algorithm
- This is the actions performed for a **single user and Task 1**
- **Double load** and then multiply as shown
- **Bottleneck** in architecture (show that they can work independantly at the same time, but floating point unit must wait for data to be ready)
- We seek to remedy this by using **column access** instead

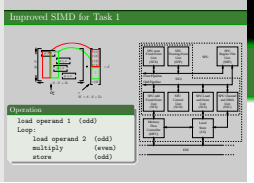
Project SIMD for Task 1





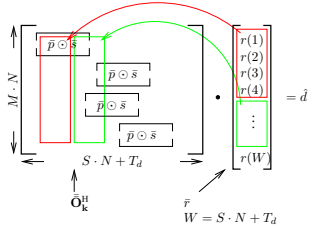
Operation

```
Loop:
  load operand 1   (odd)
  load operand 2   (odd)
  multiply         (even)
  store            (odd)
```



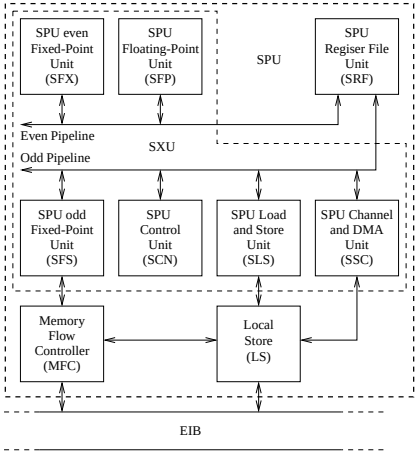
- keep **one operand** in register file
- **Reuse** r
- iterate over **columns**
- more **efficient load of data** ⇒ higher performance
- **Multiplies by zero**
- **double** multiplications, but better utilization of memory
- if loops are small, skip the store and keep data in **register file** (128 4-number entries), not done

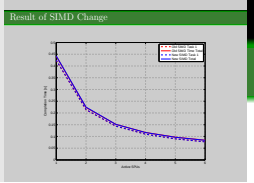
Improved SIMD for Task 1



Operation

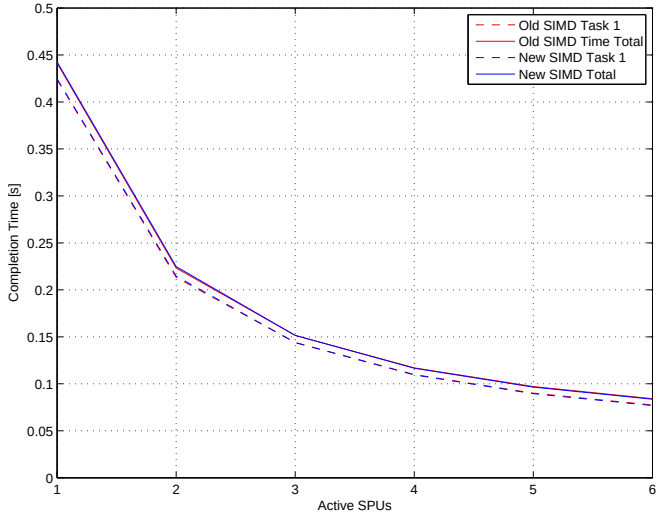
```
load operand 1  (odd)
Loop:
    load operand 2  (odd)
    multiply        (even)
    store           (odd)
```





- They are **red and blue!**
- Multiplies by **zero**
- **Double** as many multiplications, achieves same time (overlapped)
- Definitely the **way to go** for further work
- Especially if the multiply by **zeroes can be avoided** somehow

Result of SIMD Change

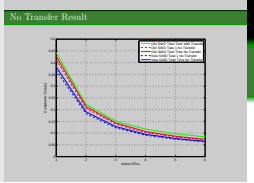


Performance Measure with no Data Transfers
Measurement
Removed all data transfers from code, except:
Initializations
Transfer of $\bar{D}$ for each user
Does this yield a significant improvement?
Does this have any effect on the improved SIMD implementation?

Performance Measure with no Data Transfers

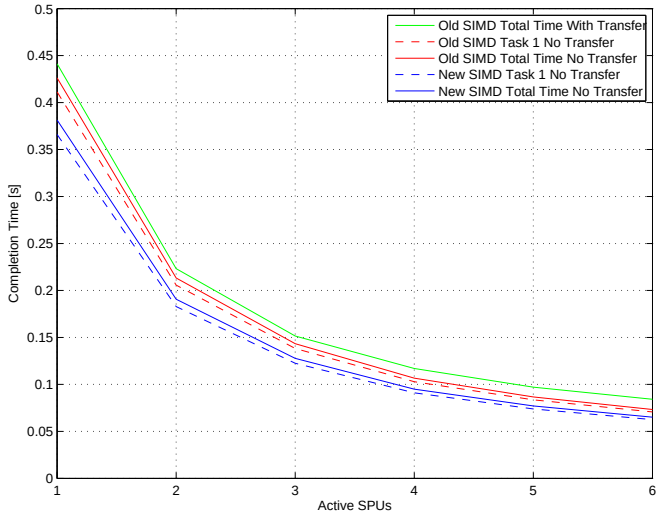
Measurement

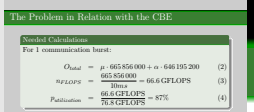
- Removed all data transfers from code, except:
  - Initializations
  - Transfer of $\bar{D}$ for each user
- Does this yield a significant improvement?
- Does this have any effect on the improved SIMD implementation?



No Transfer Result

- Describe graph **components**
- A small improvement in time (green and red)
- **Overhead and bottleneck tendency**, as the improvement is almost constant, but slight tendency to increase
- Memory **bottleneck exists** for new simd implementation as the graphs are no longer overlapped (red and blue).
- Seems like possible performance gain from new SIMD implementation is **lost** due to smaller memory transfers (it only transfers 128 values of $\bar{r}$ per computation cycle).





- **Total** amount of calculations leads to utilization for 1 burst
- **FLOPS** is only multiplications (Add is free)
- It is **possible** to solve the problem on the CBE, **computationally wise**.
- But it must be a highly optimized solution
  - Maintain **concurrency**
  - Use SPU the architecture optimally (shown in experiment), **SIMD**
  - Avoid **multiply by zeroes**
- Problem **size** changes **(transition)**
- Model for next iteration is shown next

The Problem in Relation with the CBE

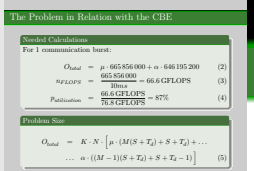
Needed Calculations

For 1 communication burst:

$$O_{total} = \mu \cdot 665\,856\,000 + \alpha \cdot 646\,195\,200 \quad (2)$$

$$n_{FLOPS} = \frac{665\,856\,000}{10ms} = 66.6\,\text{GFLOPS} \quad (3)$$

$$p_{utilization} = \frac{66.6\,\text{GFLOPS}}{76.8\,\text{GFLOPS}} = 87\% \quad (4)$$



- **Total** amount of calculations leads to utilization for 1 burst
- **FLOPS** is only multiplications (Add is free)
- It is **possible** to solve the problem on the CBE, **computationally wise**.
- But it must be a highly optimized solution
  - Maintain **concurrency**
  - Use SPU the architecture optimally (shown in experiment), **SIMD**
  - Avoid **multiply by zeroes**
- Problem **size** changes **(transition)**
- Model for next iteration is shown next

The Problem in Relation with the CBE

Needed Calculations

For 1 communication burst:

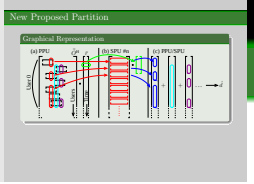
$$O_{total} = \mu \cdot 665\,856\,000 + \alpha \cdot 646\,195\,200 \quad (2)$$

$$n_{FLOPS} = \frac{665\,856\,000}{10ms} = 66.6\,\text{GFLOPS} \quad (3)$$

$$p_{utilization} = \frac{66.6\,\text{GFLOPS}}{76.8\,\text{GFLOPS}} = 87\% \quad (4)$$

Problem Size

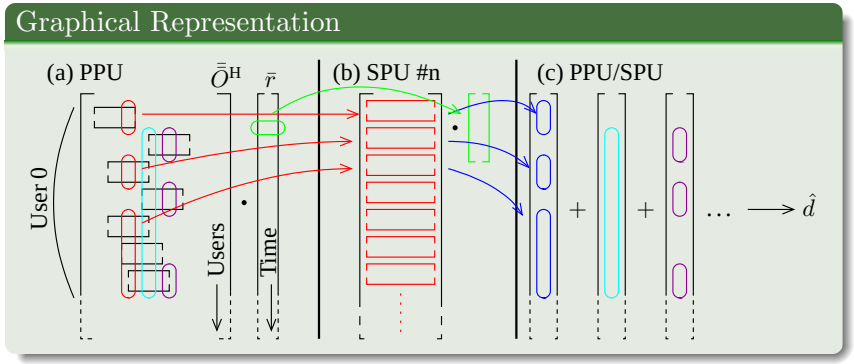
$$O_{total} = K \cdot N \cdot \Big[\mu \cdot (M(S+T_d)+S+T_d)+\ldots$$
$$\ldots\ \alpha \cdot ((M-1)(S+T_d)+S+T_d-1)\Big] \quad (5)$$

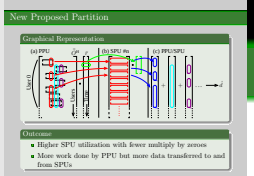


New Proposed Partition

- **Task 1** as this will yield the most significant improvement.
- SIMD operation, keep **one operand (r)** in register, load others as needed
- Performs in **time division** instead of user division
- Non-zero entries from Oh **extracted by PPU** (control structure) and sent to SPU for column-wise computation.
- **Generate spreading, scrambling on PPU** or in **HW**. SPU treats Oh as unknown data seen from SPU.
- Multiplication performed **concurrently**. Data for **summation** in (c) could come from same SPU, different SPU and be performed both on SPU or on PPU.
- Write in **assembler** as the compiler output isn't fast enough
- Conclude on this (**transition**)

Graphical Representation

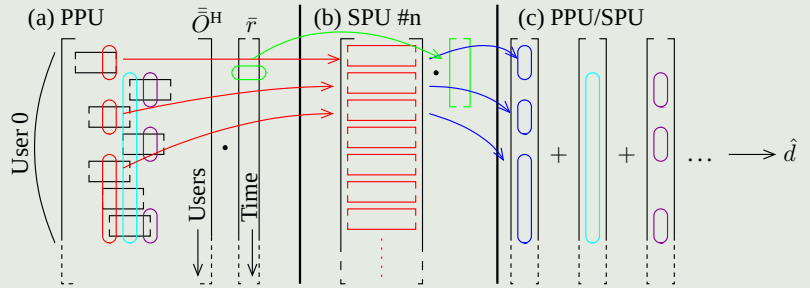




- **Task 1** as this will yield the most significant improvement.
- SIMD operation, keep **one operand (r)** in register, load others as needed
- Performs in **time division** instead of user division
- Non-zero entries from Oh **extracted by PPU** (control structure) and sent to SPU for column-wise computation.
- **Generate spreading, scrambling on PPU** or in **HW**. SPU treats Oh as unknown data seen from SPU.
- Multiplication performed **concurrently**. Data for **summation** in (c) could come from same SPU, different SPU and be performed both on SPU or on PPU.
- Write in **assembler** as the compiler output isn't fast enough
- Conclude on this (**transition**)

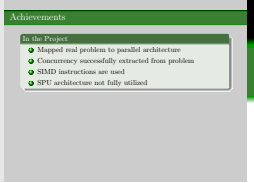
New Proposed Partition

Graphical Representation



Outcome

- Higher SPU utilization with fewer multiply by zeroes
- More work done by PPU but more data transferred to and from SPUs

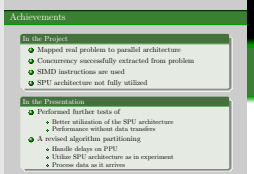


- Mapping: CDMA solution on the CBE (PS3)
- Concurrency: Signal model proposed and partitioned for concurrent execution
- SIMD: Intrinsics of signal (delays) handled so SIMD can be utilized. With a slight overhead.
- Implementation does two loads and a multiply, dictated by the partitioning

Achievements

In the Project

1. Mapped real problem to parallel architecture
2. Concurrency successfully extracted from problem
3. SIMD instructions are used
4. SPU architecture not fully utilized



- Test - SPU architecture: Perform **single load and multiply**. Implementation does **double** as many calculations (mult by zero) but achieves same time.
- Test - No transfer: Small improvement in both implementations. Seems like **overhead** in project, but **bottleneck** in improved SIMD.
- Revised Partition: Use SIMD and architecture while avoiding multiply by zeroes ⇒ handle **delays on PPU** (control structure) and do more data transfers
- Allows for **time division** (not selected in project)

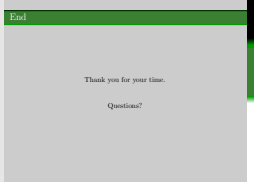
Achievements

In the Project

1. Mapped real problem to parallel architecture
2. Concurrency successfully extracted from problem
3. SIMD instructions are used
4. SPU architecture not fully utilized

In the Presentation

5. Performed further tests of
   - Better utilization of the SPU architecture
   - Performance without data transfers
6. A revised algorithm partitioning
   - Handle delays on PPU
   - Utilize SPU architecture as in experiment
   - Process data as it arrives



Smile

End

Thank you for your time.

Questions?