CO405H

Computing in Space with OpenSPL
Topic 16: Estimating and Implementing DFEs

Oskar Mencer **Georgi Gaydadjiev**

Department of Computing
Imperial College London

http://www.doc.ic.ac.uk/~oskar/
http://www.doc.ic.ac.uk/~georgig/

CO405H course page:	http://cc.doc.ic.ac.uk/openspl16/
WebIDE:	http://openspl.doc.ic.ac.uk
OpenSPL consortium page:	http://www.openspl.org

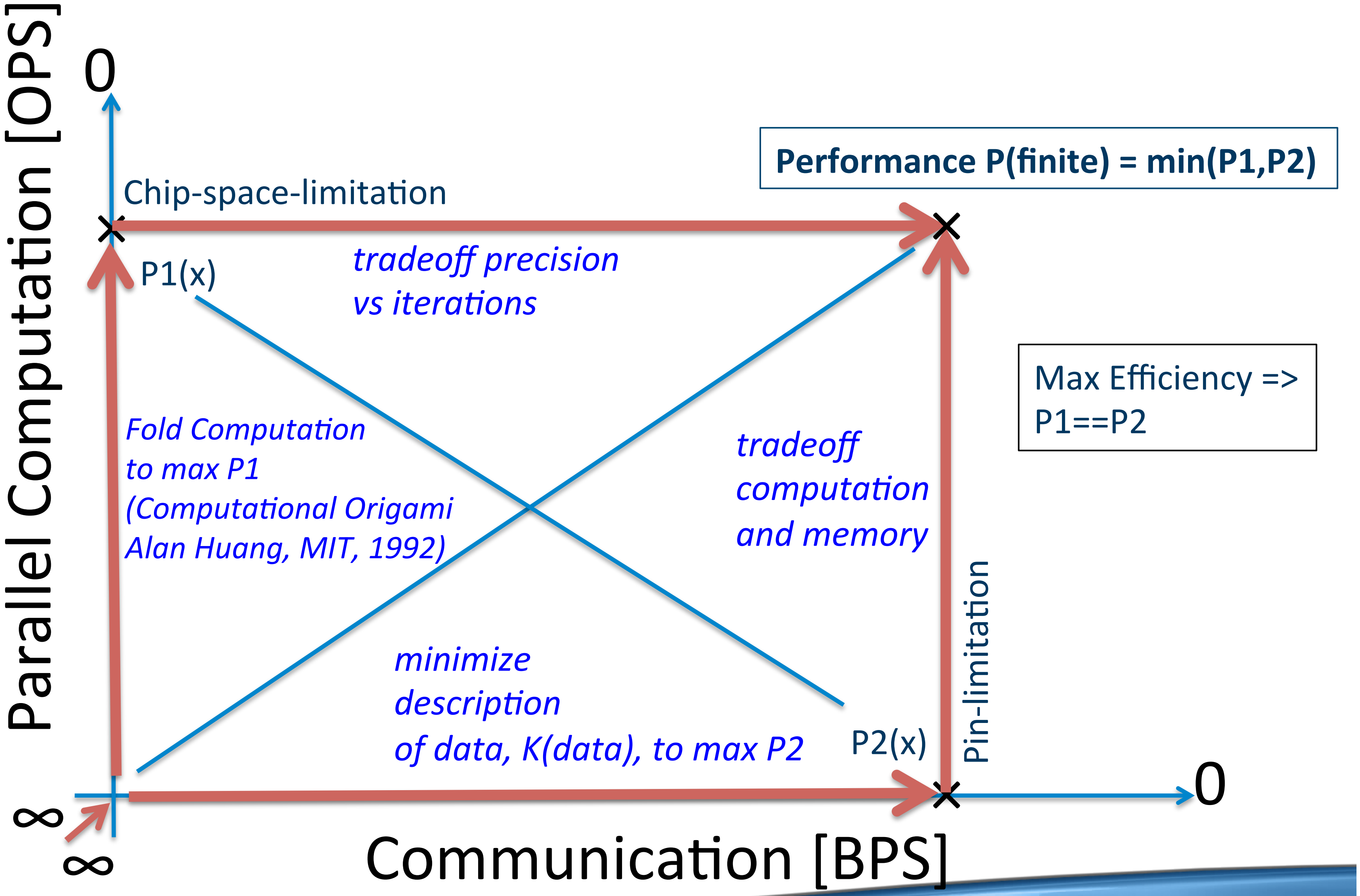
o.mencer@imperial.ac.uk g.gaydadjiev@imperial.ac.uk

Estimating DFE Performance via Modelling

Since DFEs are statically predictable, we can model throughput and computation in a simple, static way, and predict with very high accuracy how long a DFE architecture option will run to process a certain amount of data.

1. Start with Assuming there is infinite bandwidth in&out of the DFE, and infinite compute capability
2. Evaluate the case where either bandwidth or compute capability are limited
3. Evaluate the case with both bandwidth and compute capability limited to the actual case…

Estimating DFE Performance



Parallel Computation [OPS]

0

Chip-space-limitation

Performance P(finite) = min(P1,P2)

P1(x)

tradeoff precision vs iterations

Max Efficiency => P1==P2

Fold Computation to max P1 (Computational Origami Alan Huang, MIT, 1992)

tradeoff computation and memory

Pin-limitation

minimize description of data, K(data), to max P2

P2(x)

∞

∞

0

Communication [BPS]

Analysis: Step 4 – Modelling Performance

- Measure T_{OLD} = CPU time for original implementation
- Measure $T_{NON\ ACCEL}$ = time for what is being left on the CPU
- Estimate speedup if IO bound:
  - Calculate volume of data going over CPU⇔DFE bus and LMEM bus
  - Calculate $T_{IO} = \mathrm{MAX}(T_{LMEM}, T_{PCIE})$
- Estimate speedup if compute bound:
  - Resources should be roughly: *Overhead* + *NumPipes* x *PipeCost*
  - Calculate maximum number of pipes possible on DFE
  - Calculate $T_{COMPUTE} = \mathit{Ticks} / (\mathit{NumPipes} \times \mathit{Freq})$
- Compute total speedup:
  - $\mathit{Speedup} = T_{OLD} / (T_{NON\_ACCEL} + \mathrm{MAX}(T_{COMPUTE}, T_{IO}))$

Analysis: Step 4 – Modelling Performance – Option 3

- T_{OLD} = **S×85s**
- $T_{NON\ ACCEL}$ = **0s**
- Estimate speedup if IO bound:
  - T_{PCIE} = **0.00045s**
  - $T_{IO} = T_{PCIE}$ = **0.00045s**
- Estimate speedup if compute bound:
  - Resources should be roughly: *Overhead* + *NumPipes* x *PipeCost*
  - *NumPipes* = **30**, Freq = 175MHz (for MAX3, could be 100-200MHz)
  - Calculate $T_{COMPUTE} = Ticks/\ (NumPipes \times Freq)$ = **S×1.54s**
- Compute total speedup:
  - $Speedup = T_{OLD}\ /\ (T_{NON\_ACCEL} + \mathrm{MAX}(T_{COMPUTE}, T_{IO}))$ = **55.1x**

Analysis: Step 4 – Modelling Performance – Option 4

- T_{OLD} = **S×85s**
- $T_{NON\ ACCEL}$ = **S×0.00045s**
- Estimate speedup if IO bound:
  - T_{PCIE} = **S×0.0005s**
  - $T_{IO} = T_{PCIE}$ = **S×0.0005s**
- Estimate speedup if compute bound:
  - Resources should be roughly: *Overhead* + *NumPipes* x *PipeCost*
  - *NumPipes* = **32**, Freq = 175MHz
  - Calculate $T_{COMPUTE}$ = *Ticks/* (*NumPipes* x *Freq*) = **S×1.44s**
- Compute total speedup:
  - *Speedup* = T_{OLD} / (T_{NON_ACCEL} + MAX($T_{COMPUTE}$, T_{IO})) = **58.8x**

Dimensions of DFE Implementation

Implementation has three dimensions

1. Level
   - C model
   - Simulation
   - DFE
2. Precision
   - Full precision
   - Optimal precision
3. Complexity
   - Single kernel
   - All kernels

DFE Testing

- Aim for at least two tests: quick and long
- For compilation, C model is the fastest, followed by simulation, while creating DFE files can take many hours!
- When running tests, the C model will be in general faster than simulation, and DFEs are of course the fastest!
- Each level has different preferred tests that will maximise your development speed

	Small dataset (quick)	Large dataset (long)
C model	✓	✓
Simulation	✓	✘
DFE	✘	✓

- Developing and compilation is also the fastest in the C model, then in simulation, and finally on the DFE.

Dimensions of DFE Implementation

DFE Implementation has three dimensions

1. Level
   - C model: A C code implementation of the kernel
   - Simulation: A maxj implementation, simulated on a CPU
   - DFE: A .max file running on a real DFE
2. Precision
   - Full precision: Typically double precision floating point
   - Optimal precision: Could be reduced floating point, enhanced floating point (i.e. quad-precision), fixed point, or mixed.
3. Complexity
   - Single kernel: each kernel implemented and tested separately
   - All kernels: the entire DFE implementation

Dimensions of DFE Implementation

Three dimensions

1. Level
    - C model
    - Simulation
    - DFE
2. Precision
    - Full precision
    - Optimal precision
3. Complexity
    - Single kernel
    - All kernels

Use the C model to validate your analysis and as a debugging tool

Dimensions of DFE Implementation

Three dimensions

1. Level
   - C model
   - Simulation
   - DFE
2. Precision
   - Full precision
   - Optimal precision
3. Complexity
   - Single kernel
   - All kernels

NOTE: C model is not the same as C implementation. You will only have a C implementation if the app you are accelerating is written in C. You will have a different C model for each of your kernels and they do not need to be fast.

Dimensions of DFE Implementation

Three dimensions

1. Level
   - C model
   - Simulation
   - DFE
2. Precision
   - Full precision
   - Optimal precision
3. Complexity
   - Single kernel
   - All kernels

Simulation allows a quick turn around between DFE code writing and testing

Dimensions of DFE Implementation

Three dimensions

1. Level
   - C model
   - Simulation
   - DFE
2. Precision
   - Full precision
   - Optimal precision
3. Complexity
   - Single kernel
   - All kernels

DFE execution is the ultimate goal. This could come later in the acceleration process.

Dimensions of DFE Implementation

Three dimensions

1. Level
   - C model
   - Simulation
   - DFE
2. Precision
   - Full precision
   - Optimal precision
3. Complexity
   - Single kernel
   - All kernels

Use full precision to test functionality. Full precision on the DFE might mean that fewer pipes can be used.

Dimensions of DFE Implementation

Three dimensions

1. Level
   - C model
   - Simulation
   - DFE
2. Precision
   - Full precision
   - Optimal precision
3. Complexity
   - Single kernel
   - All kernels

Remember that on a DFE you are not restricted to 32- and 64-bit numbers. You could for example compute in dfeFloat(7, 41) !

Dimensions of DFE Implementation

Three dimensions

1. Level
   - C model
   - Simulation
   - DFE
2. Precision
   - Full precision
   - Optimal precision
3. Complexity
   - Single kernel
   - All kernels

Finding the optimal precision might mean that you can increase the number of kernels or pipes in your algorithm

Dimensions of DFE Implementation

Three dimensions

1. Level
   - C model
   - Simulation
   - DFE
2. Precision
   - Full precision
   - Optimal precision
3. Complexity
   - Single kernel
   - All kernels

Testing your kernels separately makes it simpler to debug and decide on the optimal precision.

Dimensions of DFE Implementation

Three dimensions

1. Level
   - C model
   - Simulation
   - DFE
2. Precision
   - Full precision
   - Optimal precision
3. Complexity
   - Single kernel
   - All kernels

Test comes last:
first in your C model,
then in simulation, and
finally on the DFE.

Dimensions of DFE Implementation

Three dimensions

1.
2.
3.

Important rule: follow the "Gray code rule" during development. That is, change only one of the dimensions at a time.

For example:

- Do not go from Simulation/Full Precision/Single Kernel to **DFE**/**Optimal precision**/Single Kernel

- Instead, for example, go from Simulation/FullPrecision/SingleKernel to Simulation/**Optimal Precision**/SingleKernel to **DFE**/**Optimal Precision**/Single Kernel

…remember DFE Testing

- Aim for at least two tests: quick and long
- For compilation, C model is the fastest, followed by simulation, while creating DFE files can take many hours!
- When running tests, the C model will be in general faster than simulation, and DFEs are of course the fastest!
- Each level has different preferred tests that will maximise your development speed

	Small dataset (quick)	Large dataset (slow)
C model	✓	✓
Simulation	✓	✗
DFE	✗	✓

- Developing and compilation is also the fastest in the C model, then in simulation, and finally on the DFE.

Testing the N-body Problem

- For the N-Body problem
  - Do a 1000 particle run and a 90,000 particles run with the C model and store results!
  - Test low number of particles (~1000) in Simulation
  - Test 1000 and 90,000 particles on the DFE and compare 1000 particles to simulation and 9000 particles to the stored results of the C model…

DFE Optimisation

- Once the DFE is working with all kernels, we optimise DFE performance
- Goals of optimisations:
  - Fit more compute on DFE
  - Increase frequency of kernels
  - Reduce expensive data movements (e.g. back and forth between CPU and LMEM)

DFE Optimisation

- The four dimensions of Optimisation:
  - Parallelism
    Instantiate multiple kernels and multi-pipe inside the kernel
  - Bandwidth
    How much data can you afford to move between DFE/CPU/LMEM?
    Use encoding, compression etc to increase effective bandwidth.
  - Area
    Resource usage as shown by the IDE and build logs. Approaching 100% for each of the resources increases compile times
  - Utilisation
    Actual compute. Need to make sure that there are no bubbles in the pipelines and all stages of computation are used for real computations with real data.
  - Frequency:
    Each Kernel can have it's own clock frequency.
    Higher frequency means higher throughput.

DFE Optimisation

- These four dimensions affect each other, for example:
  - Increasing utilisation makes it harder to build at high frequency
  - Increasing frequency brings your bandwidth utilisation closer to the DFE limit since you consume data at a faster rate
  - Higher Utilisation means more data is required to feed the compute unit

Conclusions – Estimating and Implementing DFEs

- Move the compute intensive part to the DFE, as well as any other part that requires access to large amount of data created by the compute intensive part
- Develop first in your C model,
  then in Simulation and finally on the DFE
- Run small fast tests in simulations,
  and long tests on the DFE
- Fill the space on the chip to
  maximise compute per cycle
- **Do not forget the "Gray code rule":
  only change one thing at a time**

Exercises

For the questions below, assume the accelerator chip runs at 100MHz and can sustain 30GB/s bandwidth from DRAM and 2GB/s in each direction from PCIe.

1. A kernel has two inputs from DRAM (*a* and *b*) and one output (*c*) to DRAM. Input *a* reads 8GB, input *b* reads 20GB and output *c* writes 25GB. The kernel runs for 100M cycles. How long does the computation take to complete?

2. A kernel performs an iterative calculation. It iterates on a set of 40,000 data values. Each value is *32* bytes. The kernel has a single input *a* and output *b* to/from PCI Express. The kernel initially reads the starting values for the data set over 40,000 cycles, then computes over the entire set (1 value per cycle) for 10 iterations, then spends 40,000 cycles writing the results back to PCI Express. How long does the entire computation take to complete?

3. Design a kernel to implement the software loop below, model its performance assuming the array *M* is held in DRAM and the arrays *p* and *q* are stored in host memory (accessed via PCIe).

```
for (int i = 0; i < 1000000; i++) {
    q[i] += p[i] * ((M[i+1] + M[i] + M[i-1])/2);
}
```