Scaling the Peak: Maximizing floating point performance on the Epiphany NoC

Anish Varghese, Gaurav Mitra, Robert Edwards and Alistair Rendell

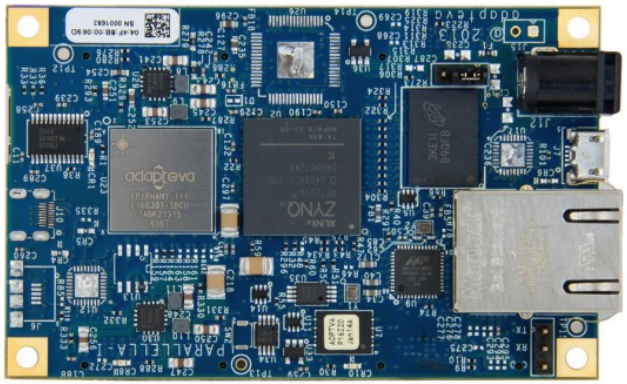
Research School of Computer Science
The Australian National University

May 30, 2015

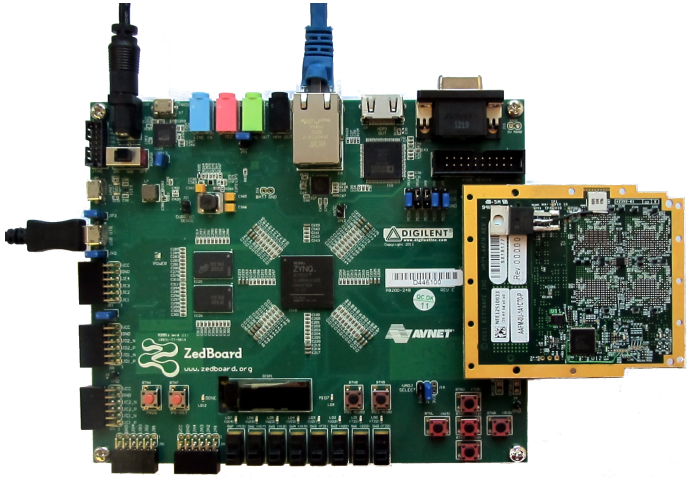
Outline

Energy efficient HPC at ANU

Parallella-16



Parallella-64 Prototype

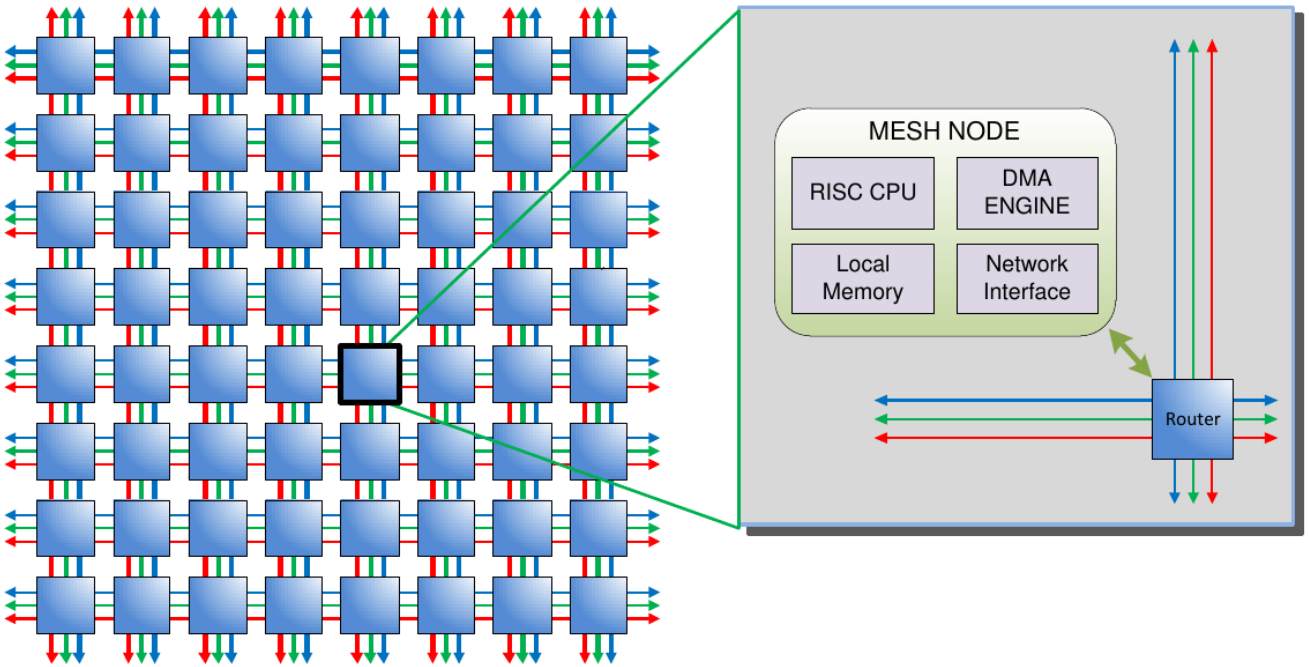


NVIDIA Jetson TK1 - ARM-GPU SoC

TI Keystone II - ARM-DSP SoC

Outline

Programming Considerations



- Store code and data in different banks
- Interleave FMADD with load/stores
- Unroll inner loops

Outline

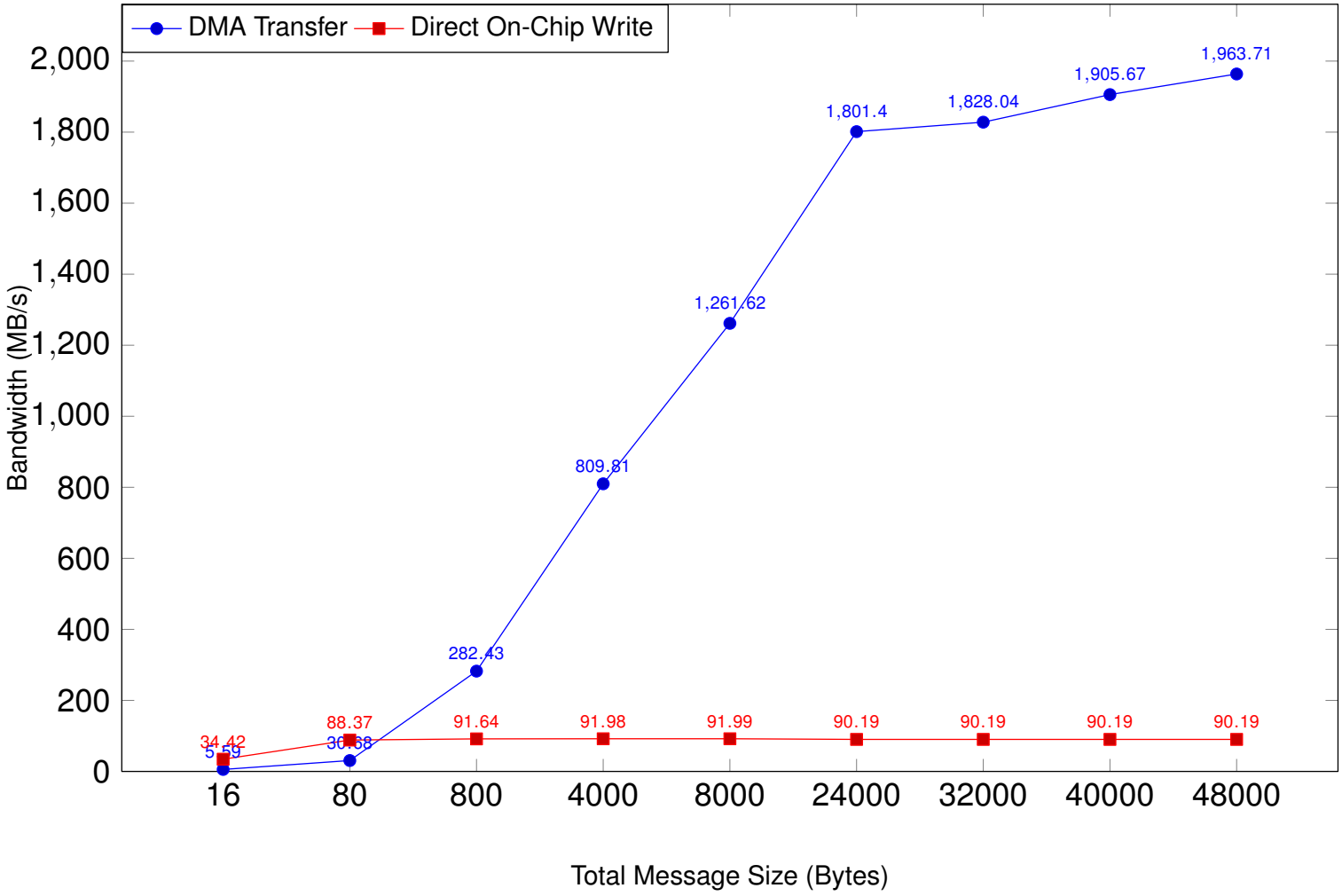
Experiment Platform

Parallella-64 Prototype

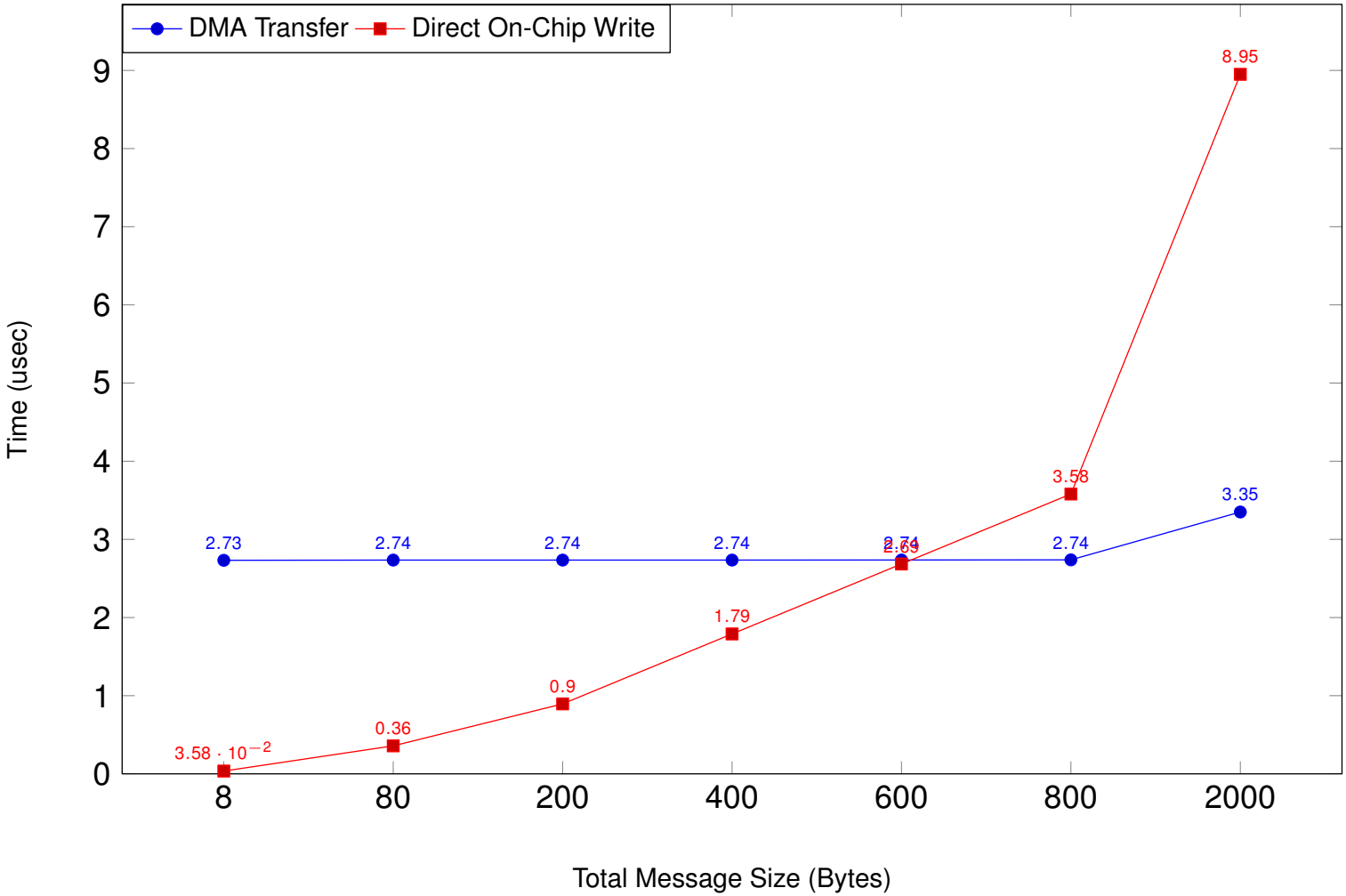
- ZedBoard evaluation module with Zynq SoC
- Daughter card with Epiphany-IV 64-core (E64G401)
- Dual core ARM Cortex-A9 host at 667 MHz
- Epiphany eCores at 600 MHz
- 512 MB of DDR3 RAM on the host
- 32 MB shared with eCores

Outline

DMA vs Direct Write Bandwidth



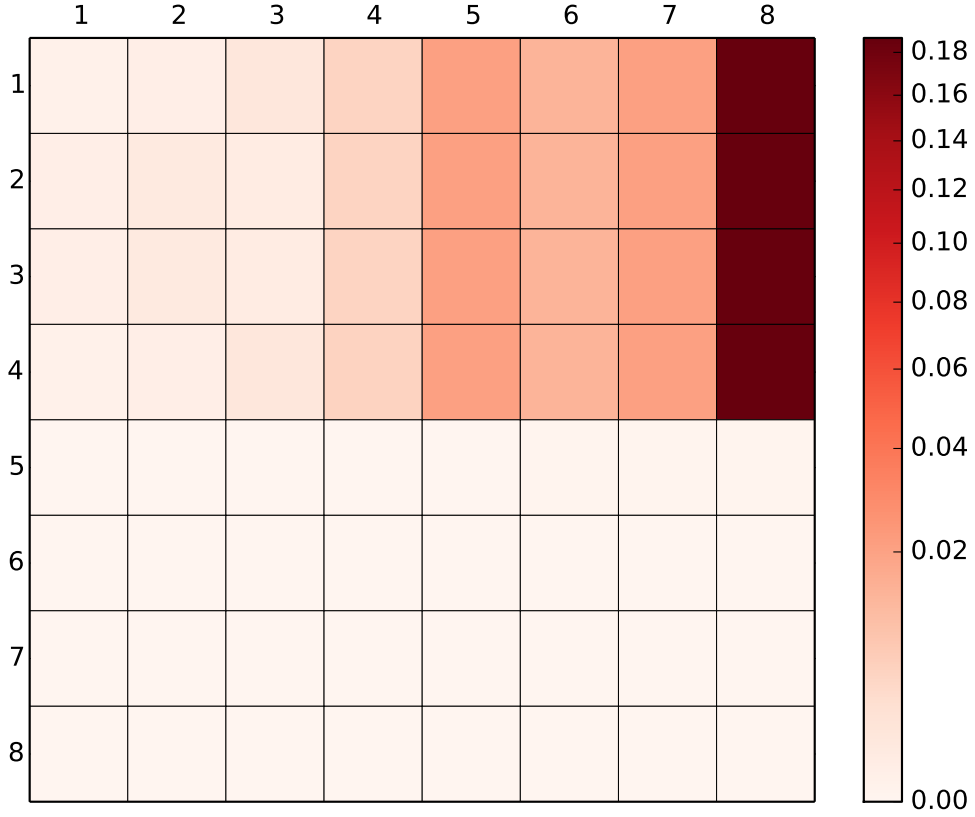
Message Transfer Latency



- Latency of ≈ 7 cycles per transfer

Outline

Core-wise utilization of external memory link



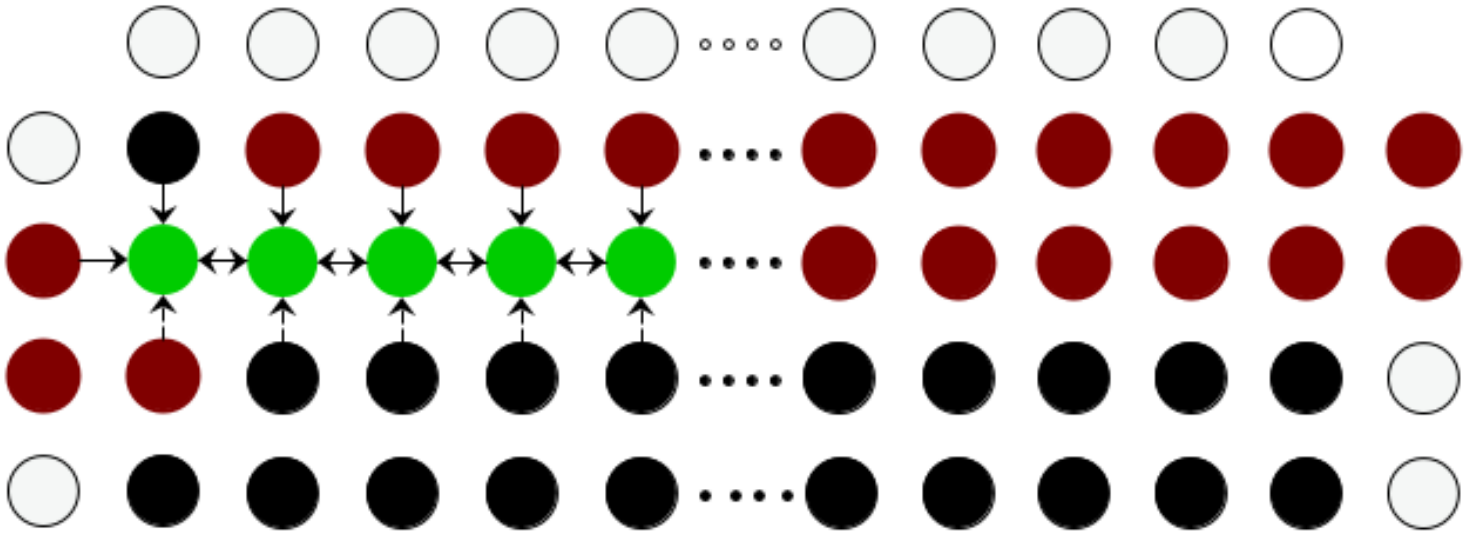
- 150 MB/sec using Direct Writes
- 400 MB/sec using DMA

Outline

Five-point star-shaped heat stencil

$$
\begin{aligned}
T_{new_{i,j}} = {} & w_1 * T_{prev_{i,j+1}} + w_2 * T_{prev_{i,j}} \\
& + w_3 * T_{prev_{i,j-1}} + w_4 * T_{prev_{i+1,j}} \\
& + w_5 * T_{prev_{i-1,j}}
\end{aligned}
$$

Single-Core Stencil - 20 columns



Diving into Assembly

```
.Lb:
dogrid1 26,27,28,29,30,47,48,49,50,51,52,53, 1, 2, 3, 4, 5, 6, 7, 8, 9,10
ldr r20,[r0,#0 + stride]
dogrid0 31,32,33,34,35,52,53,54,55,56,57,58, 6, 7, 8, 9,10,11,12,13,14,15
ldr r41,[r1,#0]
dogrid1 36,37,38,39,40,57,58,59,60,61,62,63,11,12,13,14,15,16,17,18,19,20
str r21,[r1],#stride
; 2nd row
dogrid0 43,44,45,46,47,20,21,22,23,24,25,26,16,17,18,19,20,1+stride,2+stride,3+stride
      ,4+stride,5+stride
add r0,r0,#(stride * 4)
dogrid1 48,49,50,51,52,25,26,27,28,29,30,31, 1, 2, 3, 4, 5, 6, 7, 8, 9,10
ldr r42,[r0,#stride]
dogrid0 53,54,55,56,57,30,31,32,33,34,35,36, 6, 7, 8, 9,10,11,12,13,14,15
ldr r63,[r1,#0]
dogrid1 58,59,60,61,62,35,36,37,38,39,40,41,11,12,13,14,15,16,17,18,19,20
str r43,[r1],#stride
; 1st row
dogrid0 21,22,23,24,25,42,43,44,45,46,47,48,16,17,18,19,20,1+stride,2+stride,3+stride
      ,4+stride,5+stride
add r0,r0,#(stride * 4)
sub r2,r2,#1
nop
bne .Lb
```

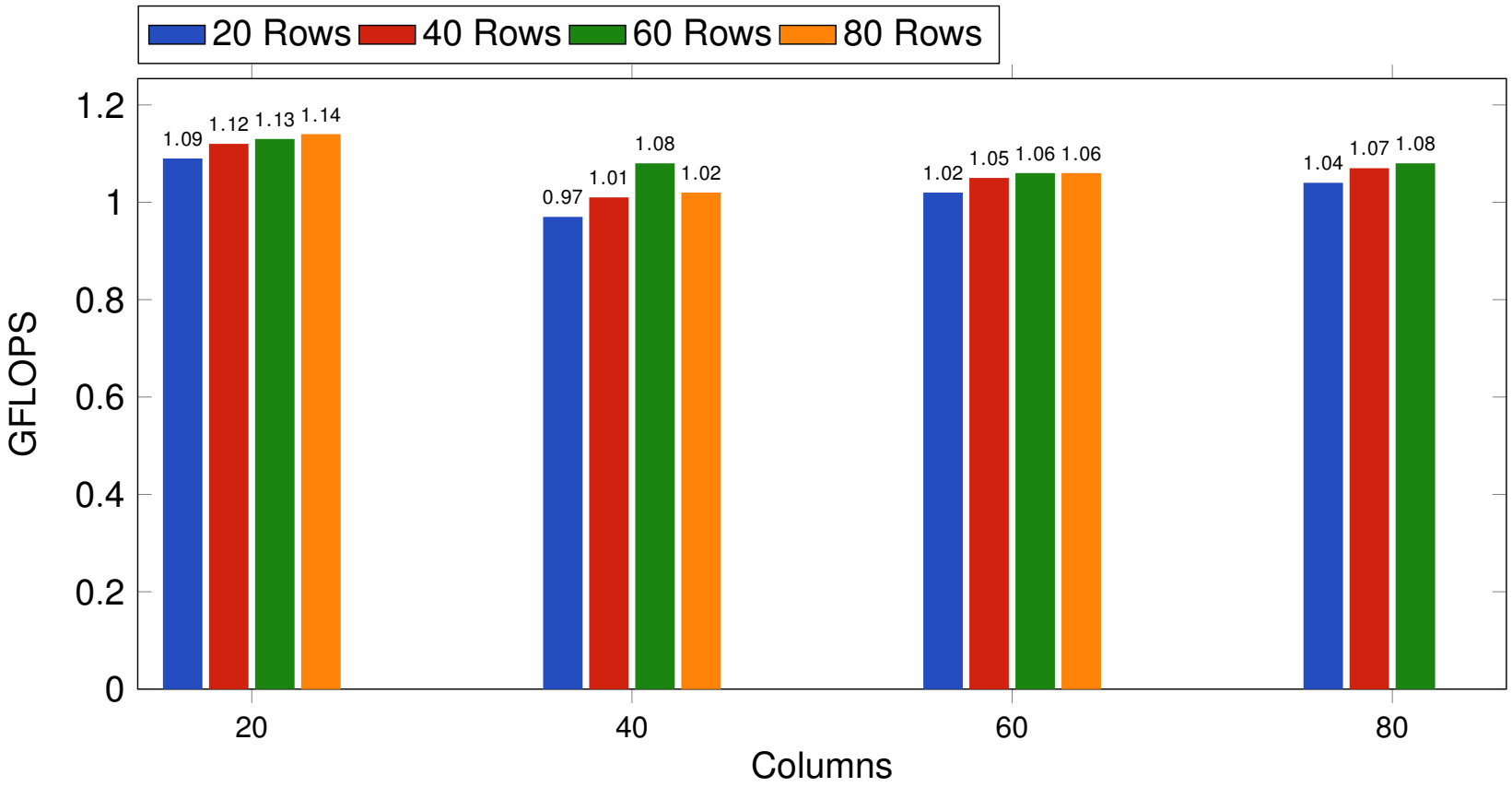
Writing Macros

```
.macro dogrid0 a0,a1,a2,a3,a4,b0,b1,b2,b3,b4,b5,b6,o0,o1,o2,o3,o4,i0,i1,i2,i3,i4
fmadd r15,r\a0,r3
str r8,[r0,#\o0]
fmadd r16,r\a1,r3
str r9,[r0,#\o1]
fmadd r17,r\a2,r3
str r10,[r0,#\o2]
fmadd r18,r\a3,r3
str r11,[r0,#\o3]
fmadd r19,r\a4,r3
str r14,[r0,#\o4]
fmadd r15,r\b0,r4
fmadd r16,r\b1,r4
fmadd r17,r\b2,r4
fmadd r18,r\b3,r4
fmadd r19,r\b4,r4
fmadd r15,r\b1,r5
.
.
.
```

Interleaved pair

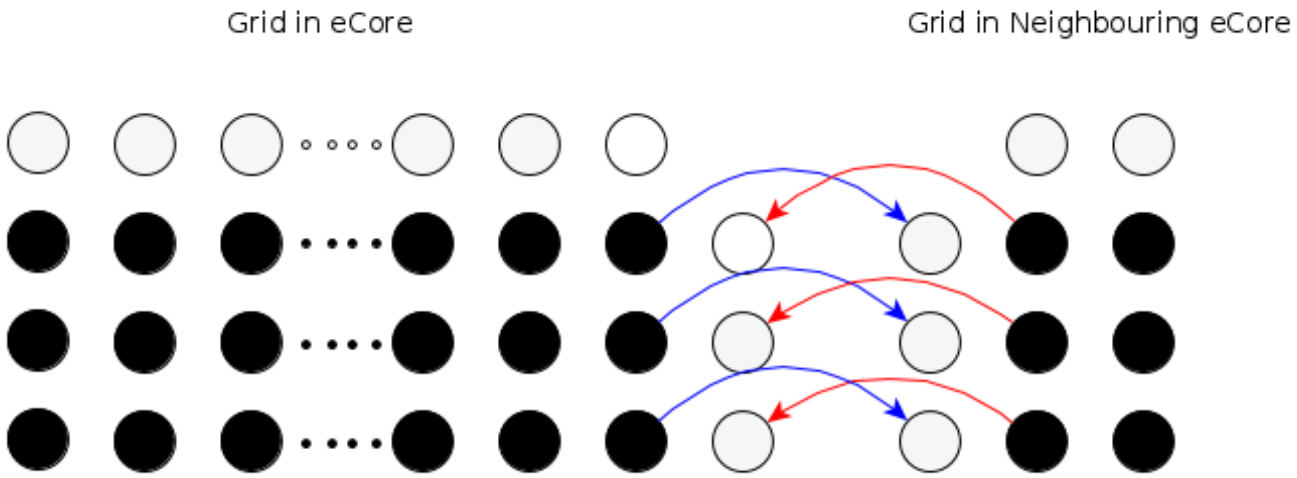
Code available at https://github.com/parallella/parallella-examples/tree/master/heat_stencil

Floating point performance - Single-Core



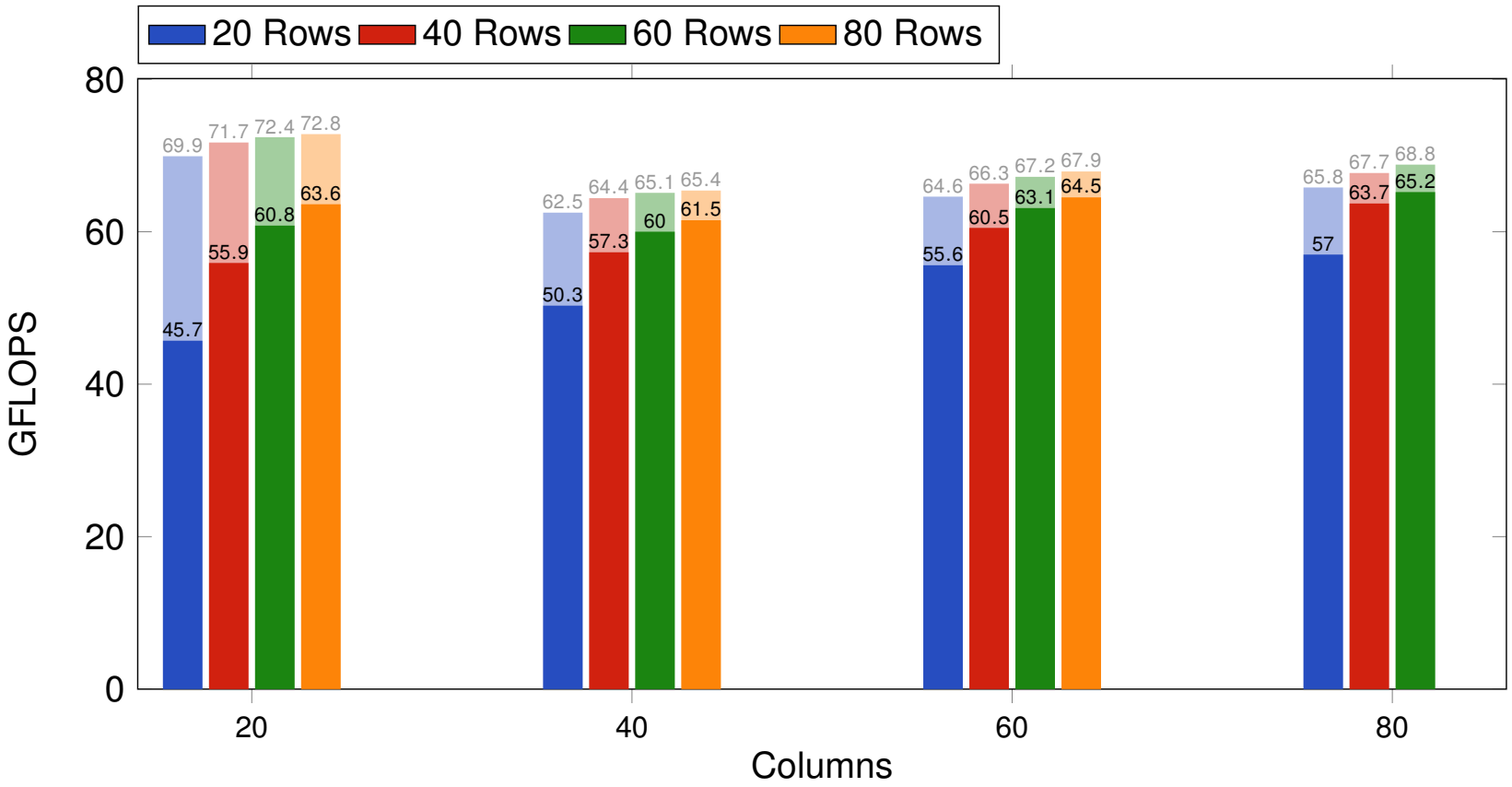
- Stencil evaluated for 50 iterations.
- 81-95% of peak performance

Multi-Core Stencil



- Synchronization between neighbouring eCores
- DMA for boundary transfers

Floating point performance - Multi-Core



- Lighter colors show performance without communication
- 65.2 GFLOPS: 85% of peak performance with communication

Outline

Single-Core Matmul

$$C_{11} = A_{11}B_{11} + A_{12}B_{21} + A_{13}B_{31} + \ldots$$

$$C_{12} = A_{11}B_{12} + A_{12}B_{22} + A_{13}B_{32} + \ldots$$

$$\vdots$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21} + A_{23}B_{31} + \ldots$$

$$C_{22} = A_{21}B_{12} + A_{22}B_{22} + A_{23}B_{32} + \ldots$$

- Operand matrices in different memory banks
- Stack moved to bottom half of bank 1
- Hand-tuned assembly code

Diving into Assembly (again)

```
.macro doMult areg,index,

fmadd r32,r\areg,r16
ldrd r22,[r1,#\index + 3]
fmadd r33,r\areg,r17
ldr r\aprev,[r0,#\inc]
fmadd r34,r\areg,r18
fmadd r35,r\areg,r19
fmadd r36,r\areg,r20
fmadd r37,r\areg,r21
ldrd r16,[r1,#\index + 4]
fmadd r38,r\areg,r22
ldrd r18,[r1,#\index + 5]
fmadd r39,r\areg,r23
ldrd r20,[r1,#\index + 6]
fmadd r40,r\areg,r16
ldrd r22,[r1,#\index + 7]
.
.
.
```

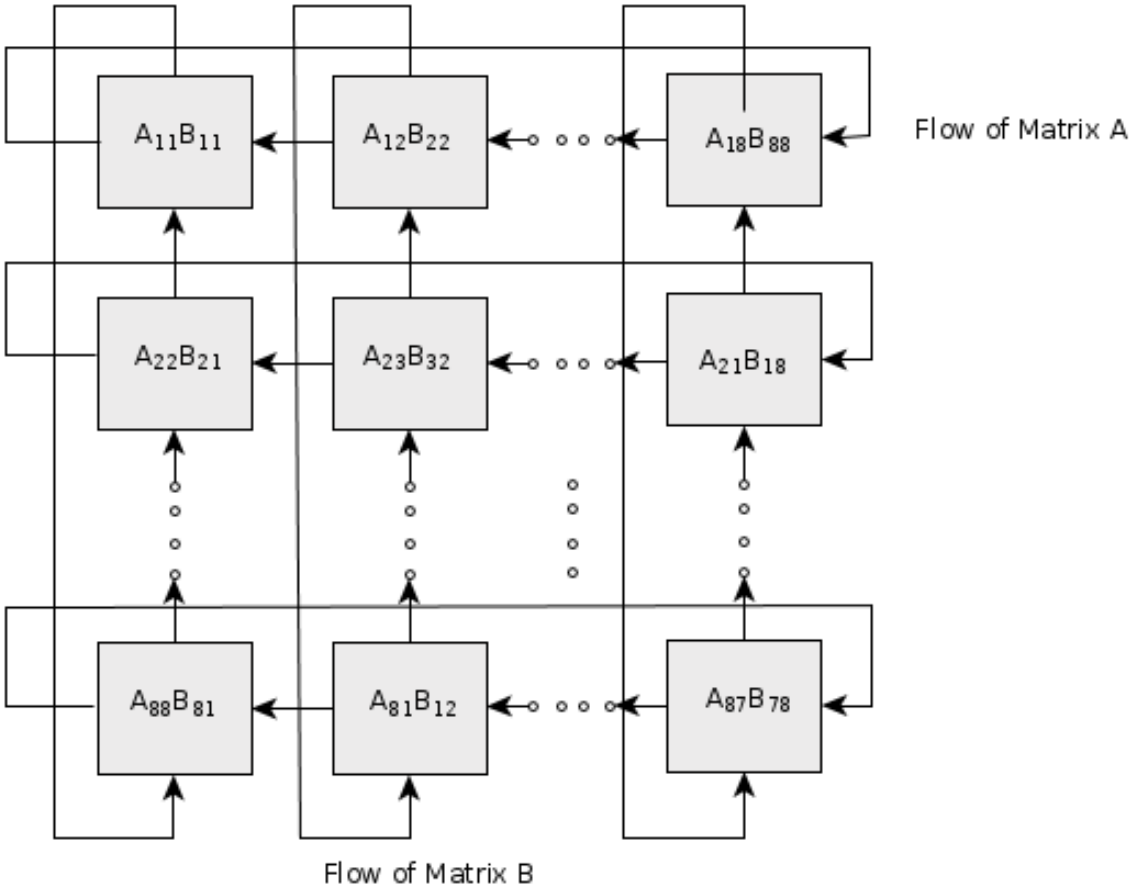
Interleaved pair

Code available at https://github.com/parallella/parallella-examples/tree/master/matmul_optimized

Floating point performance - Single-Core

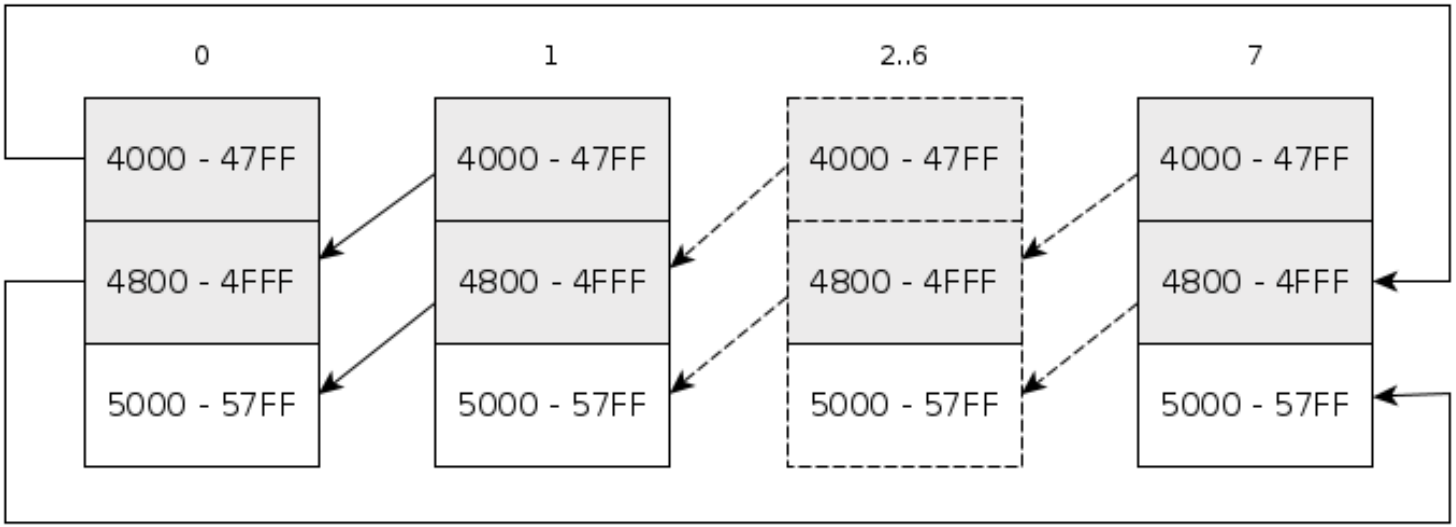
Dimensions	GFLOPS	% Peak
8×8	0.85	70.5
16×16	1.07	89.5
20×20	1.11	92.5
24×24	1.12	93.4
32×32	1.15	95.9

Multi-Core Matmul - Cannon's Algorithm



Buffering Strategy

Transfer of Matrix A and Matrix B



Floating Point Performance - Multi-Core

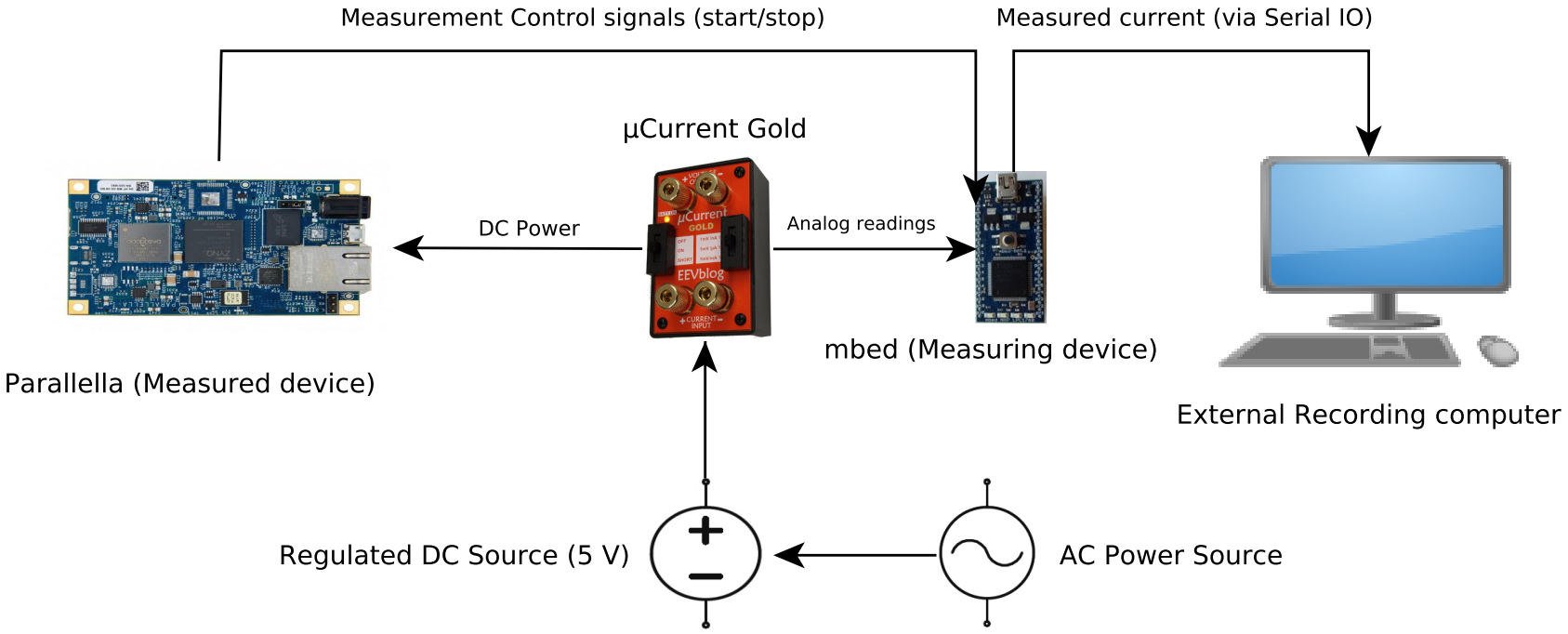
Matrix C (per-core)	Num of eCores			
	4×4		8×8	
	GFLOPS	% Peak	GFLOPS	% Peak
8×8	5.1	26	20.3	26
16×16	12.8	67	51.4	67
20×20	14.4	75	57.6	75
24×24	15.4	80	62.2	81
32×32	16.3	85	65.3	85

Floating Point Performance - Off-chip

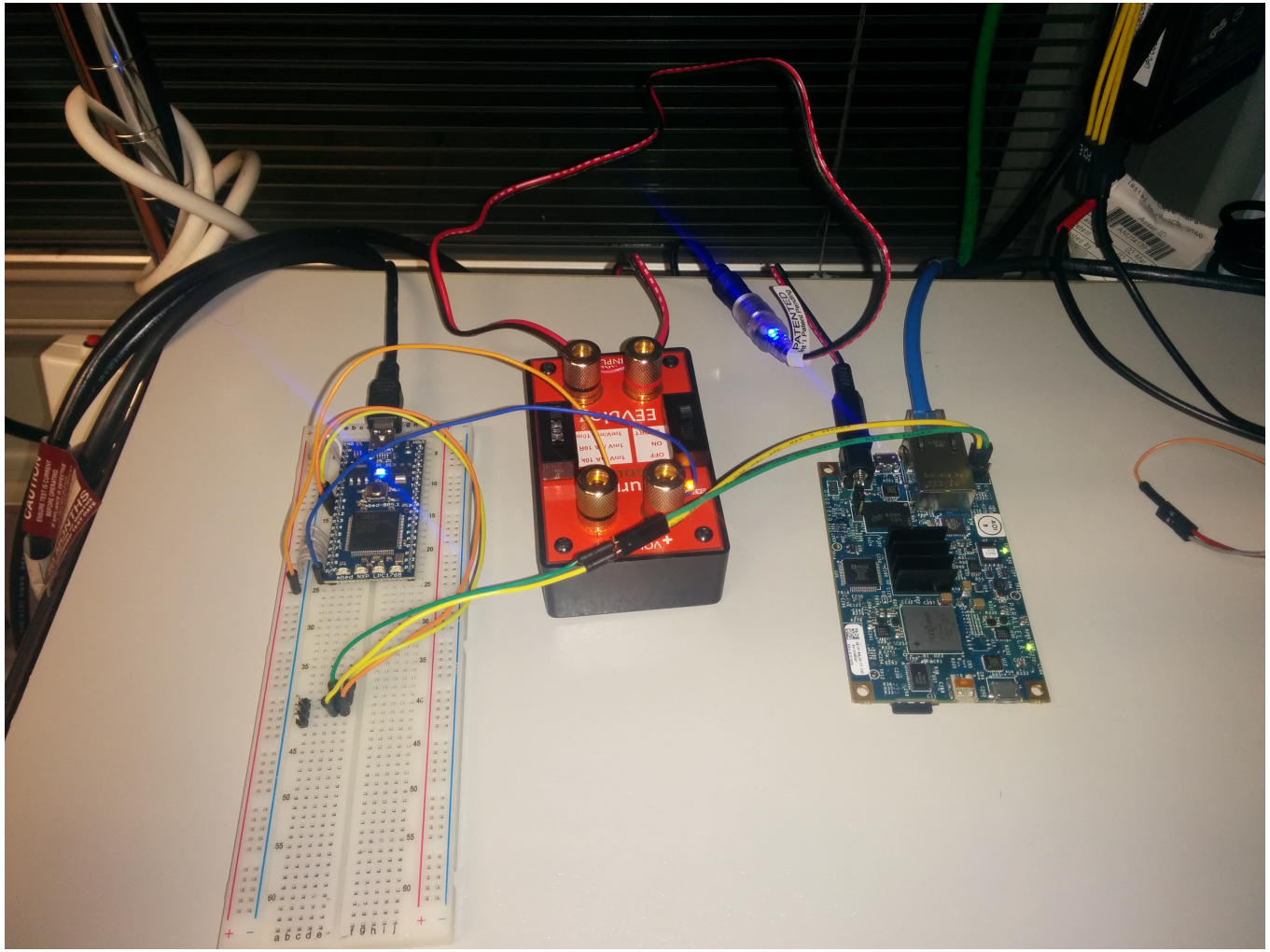
Matrix C	GFLOPS	% Peak	% Memory Transfers
512×512	8.3	10.8 %	87.2 %
1024×1024	8.6	11.1 %	86.9 %
1536×1536	6.3	8.2 %	89.1 %

Outline

Measurement Setup



Measurement Setup



Outline

Measurement API

```
/* Initialize serial port IO objects from Measured Device */
void power_init()
{
    io = new boost::asio::io_service();
    serial = new boost::asio::serial_port(*io, serial_port);
    serial->set_option(boost::asio::serial_port_base::baud_rate(baud_rate));
}

/* Send single character 'r' to reset ADC timer and start measurement */
void power_start()
{
    boost::asio::write(*serial, boost::asio::buffer("r",1));
}

/* Send single character 's' to stop ADC timer and measurement */
void power_stop()
{
    boost::asio::write(*serial, boost::asio::buffer("s",1));
}

/* Perform serial IO object cleanup */
void power_cleanup()
{
    delete io;
    delete serial;
}
```

Outline

Energy Efficiency for Matmul - Comparisons

Parallella-16

Dimensions	nJoules/Flop
128×128	0.33
256×256	1.82
512×512	1.98
1024×1024	1.83

NVIDIA Jetson

Dimensions	nJoules/Flop
128×128	9.7
256×256	1.06
512×512	0.16
1024×1024	0.06

- 128×128 experiment on Parallella-16 used only on-chip SRAM
- Same numbers can be derived analytically

Summary & Future Work

- Achieved high-performance
  - 85% of Peak for On-chip
- Introduced a method to measure energy to solution
- High programming effort was required
  - Compiler could be optimized further
- Energy efficiency
  - Observed on-chip efficiency of 330 pJoules/Flop
  - 4096 eCores => 10 pJoules/Flop
- Higher bandwidth and more SRAM per eCore needed

Questions?