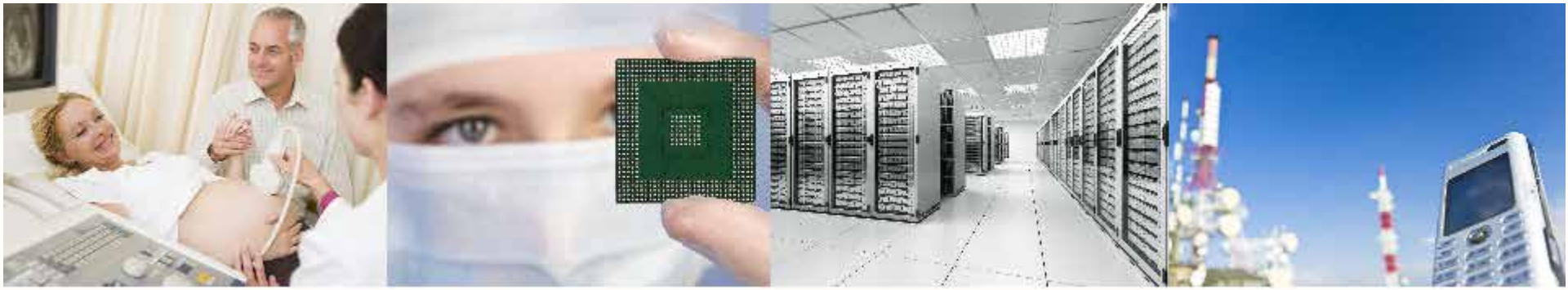


The Epiphany Manycore Architecture

Seminar at Halmstad Högskola

March 6, 2012

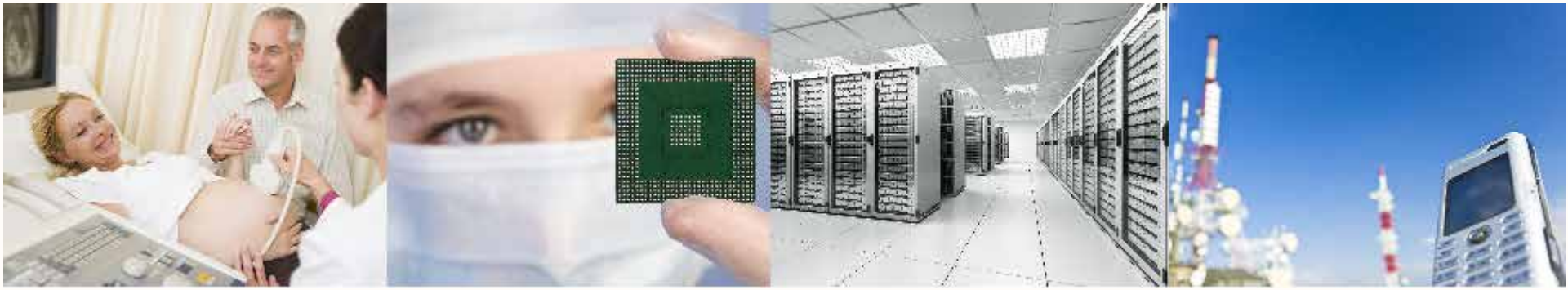
andreas@adapteva.com

+1-781-325-6688

adapteva

Adapteva Company Introduction

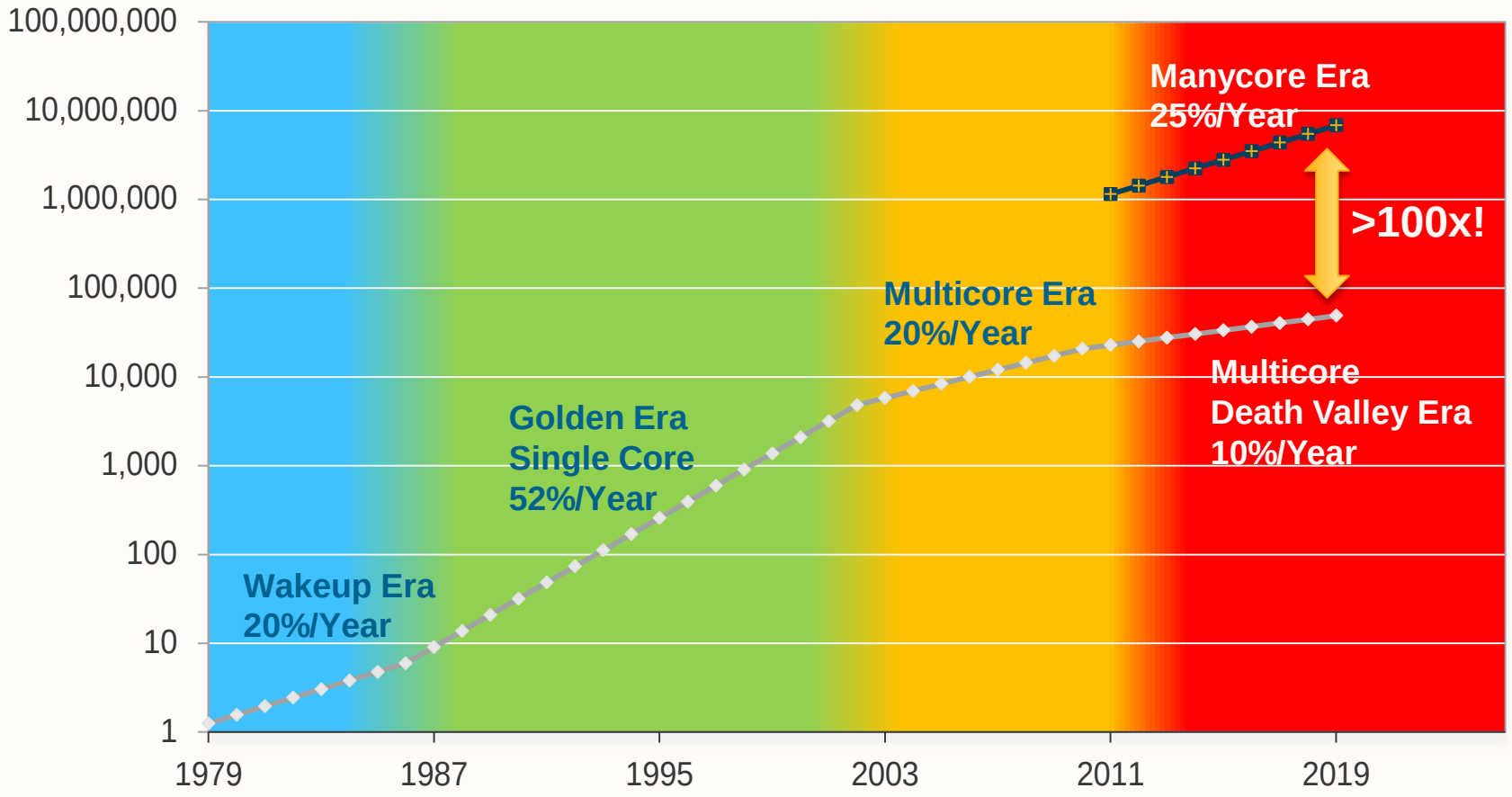
- Team:
  - Veteran low power processor design team from Analog Devices
  - Small capital efficient team with strong partnerships
- Business:
  - Founded in 2008, break even since 08/2011
  - Custom processor chips for specialty markets
  - IP business model for mobile processor market
- Technology:
  - World's most energy efficient computing platform
  - Product shipping at 65nm, sampling 28nm in Q1/2012
  - Scalable from 16 to 4096 CPU cores on a single die



Computing Philosophy

adapteva

The Wakeup Call

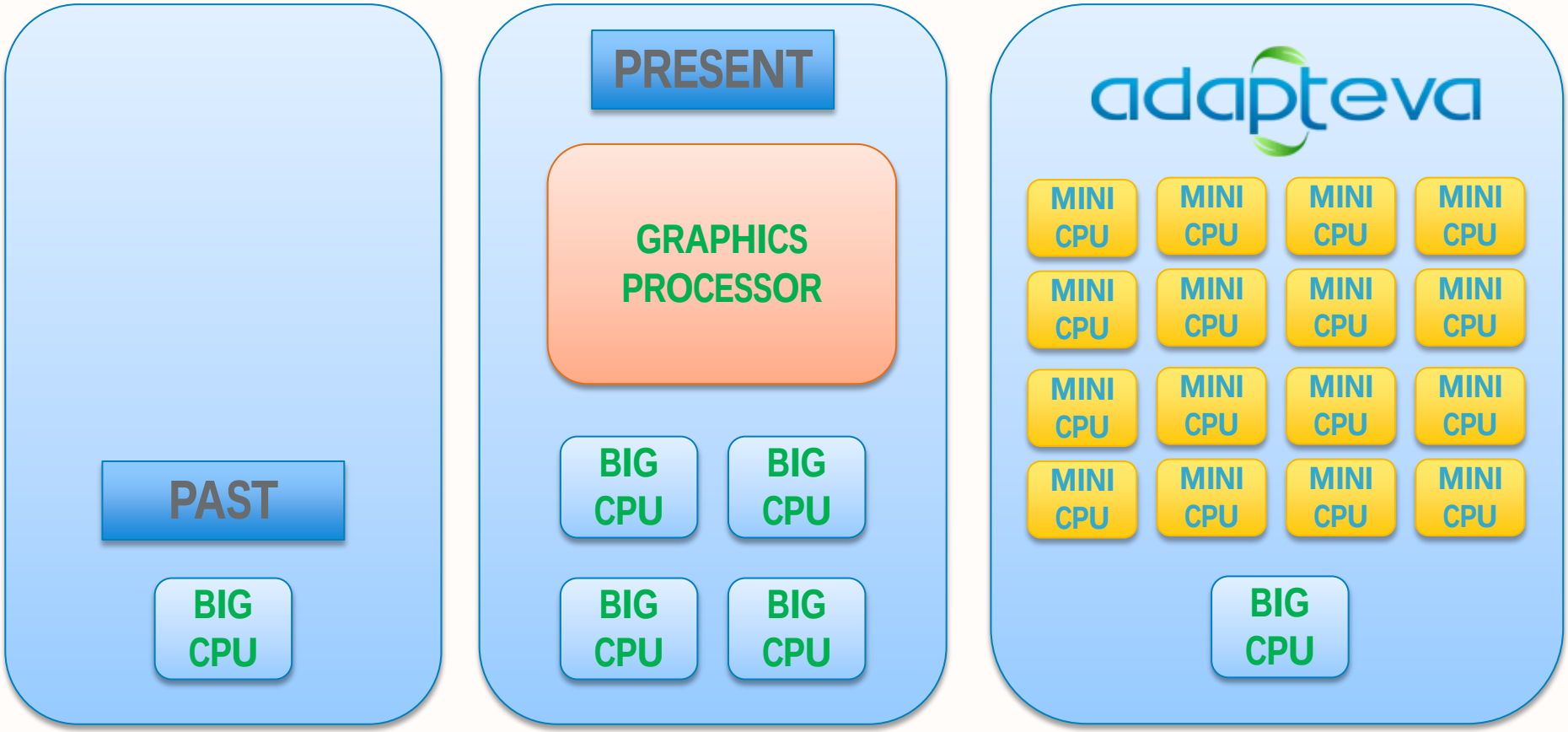


Memory Bandwidth Challenge

Programming Challenge

Scaling Challenge

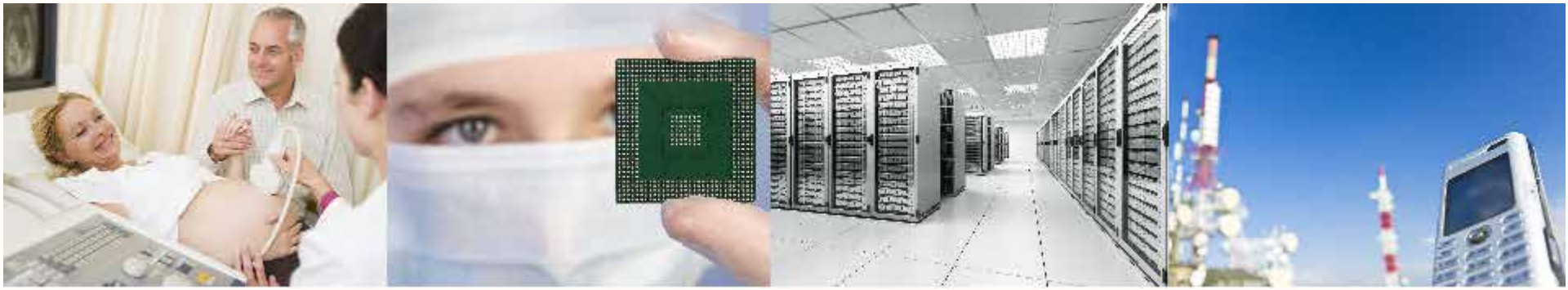
Our visions for the future of computing



Heterogeneous

Easy to Use

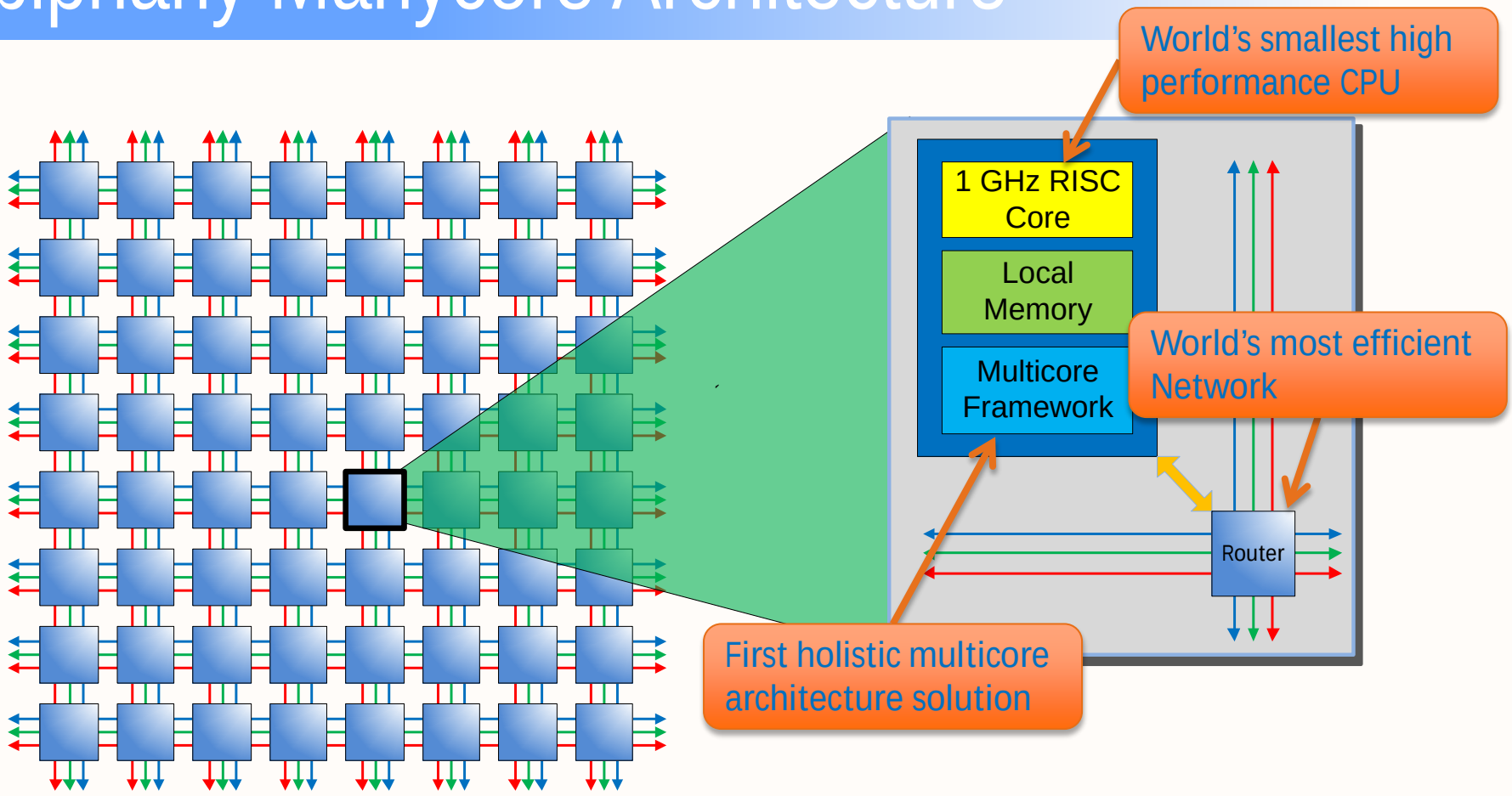
Specialized



Architecture

adapteva

Epiphany Manycore Architecture



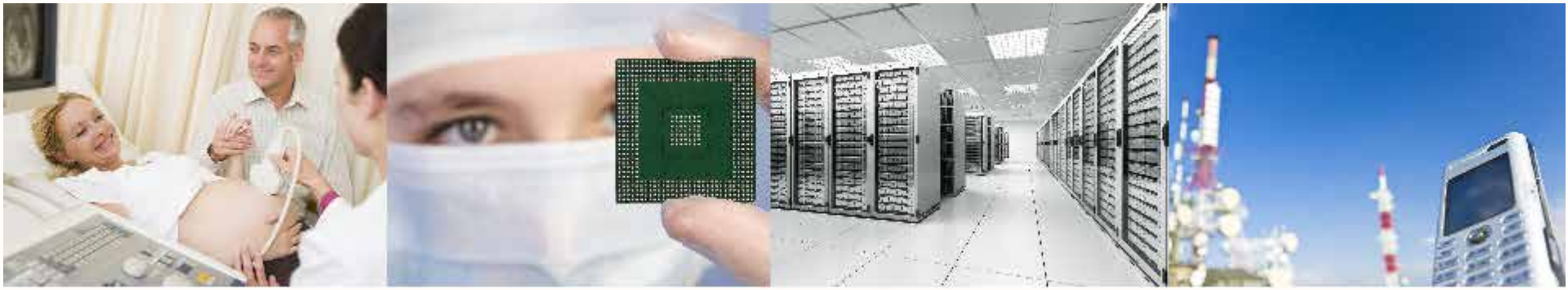
C/C++ Programmable

Scales to 4096 cores in 28nm

70 GFLOPS/Watt

Key Epiphany Features

- Each processor can run a separate and independent program (MIMD not SIMD)
- Runs ANSI-C (or Fortran?) code out of the box from SDRAM. Critical code and data should run out of core's local memory
- Optimized (but not restricted) to be a an accelerator/coprocessor
- Flat globally shared physical memory map (no L1/L2 caches)
- Row, column based address mapping
- Remote writes MUCH faster than remote reads
- Single clock cycle core to core message passing
- Native IEEE floating point support

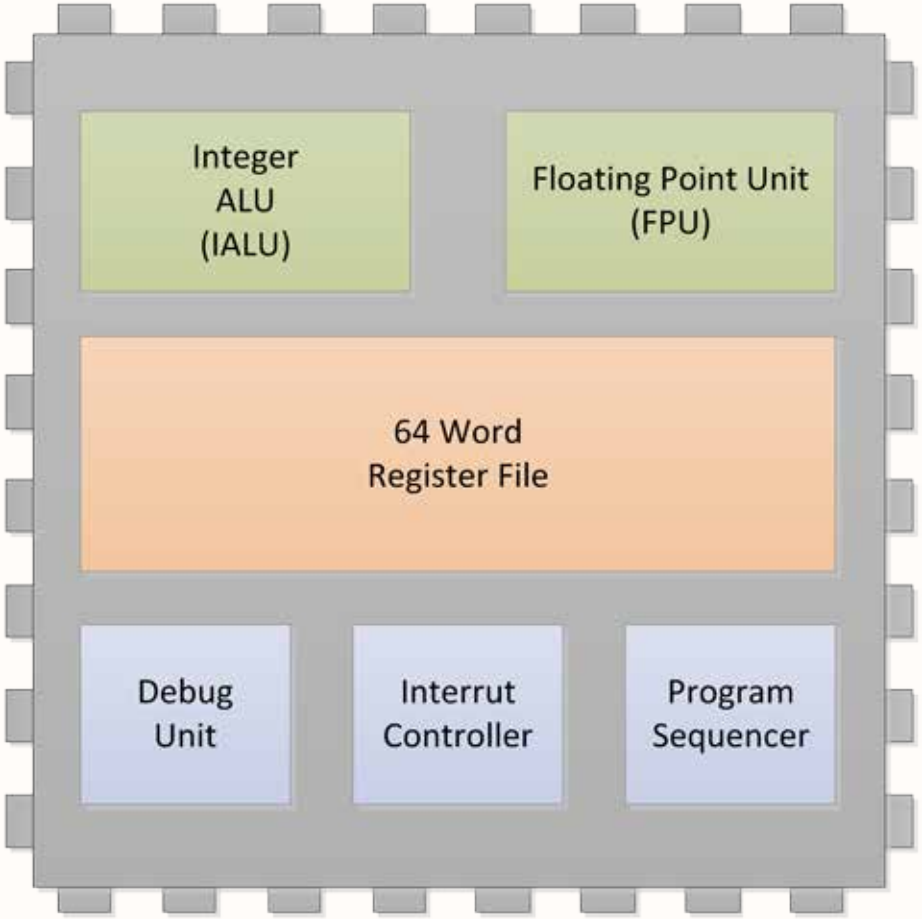


Epiphany Core

adapteva

”Mini-RISC” Processor Highlights

- Features:
  - Superscalar architecture
  - Built for multicore
  - Dense instruction encoding
  - IEEE floating point support
  - Outstanding compiler performance
- Specs:
  - 1GHz Operation at 65G
  - <25mW Peak power at 28nm
  - 0.13mm^2 total area (w/ 32KB)
- Benchmarks:
  - 1521 CoreMark score (Control)
  - <40us 1024 point FFT (DSP)
  - >90% efficiency on SGEMM (HPC)

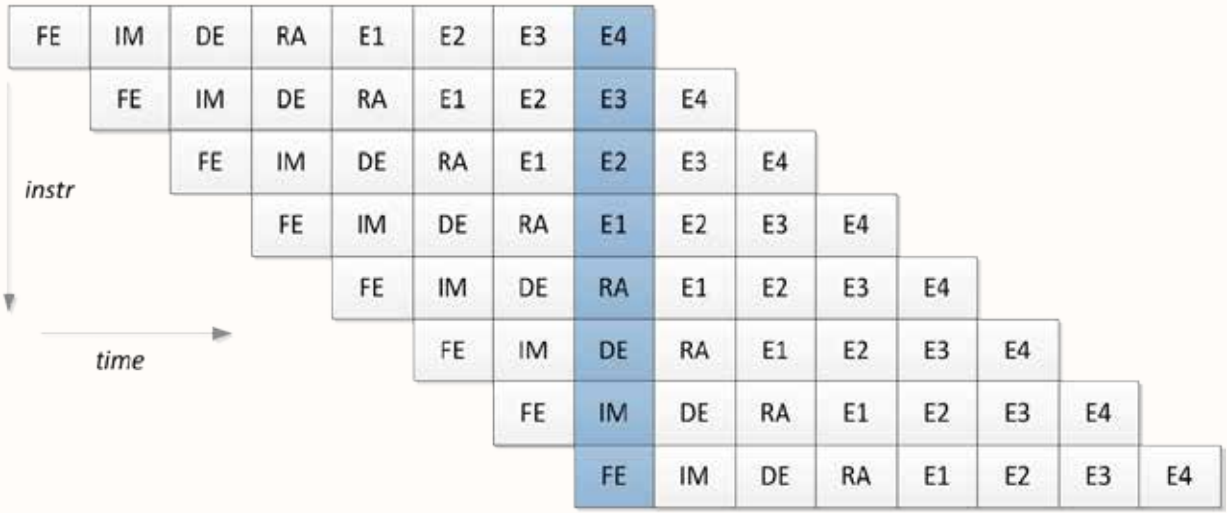


Epiphany Instruction Set (35)

- Floating Point (8)
  - FADD, FSUB, FMUL, FMADD, FMSUB, FIX, FLOAT, FABS
- Integer (9)
  - ADD, SUB, LSL, LSR, ASR, EOR, ORR, AND, BITR
- Load/Store (6)
  - (LDR,STR) * 3 addressing modes
- Branching (5)
  - B<cond>, BL, JR, JALR
- Other (7)
  - MOV<COND>, MOVT, NOP, IDLE, RTI, GID, GIE

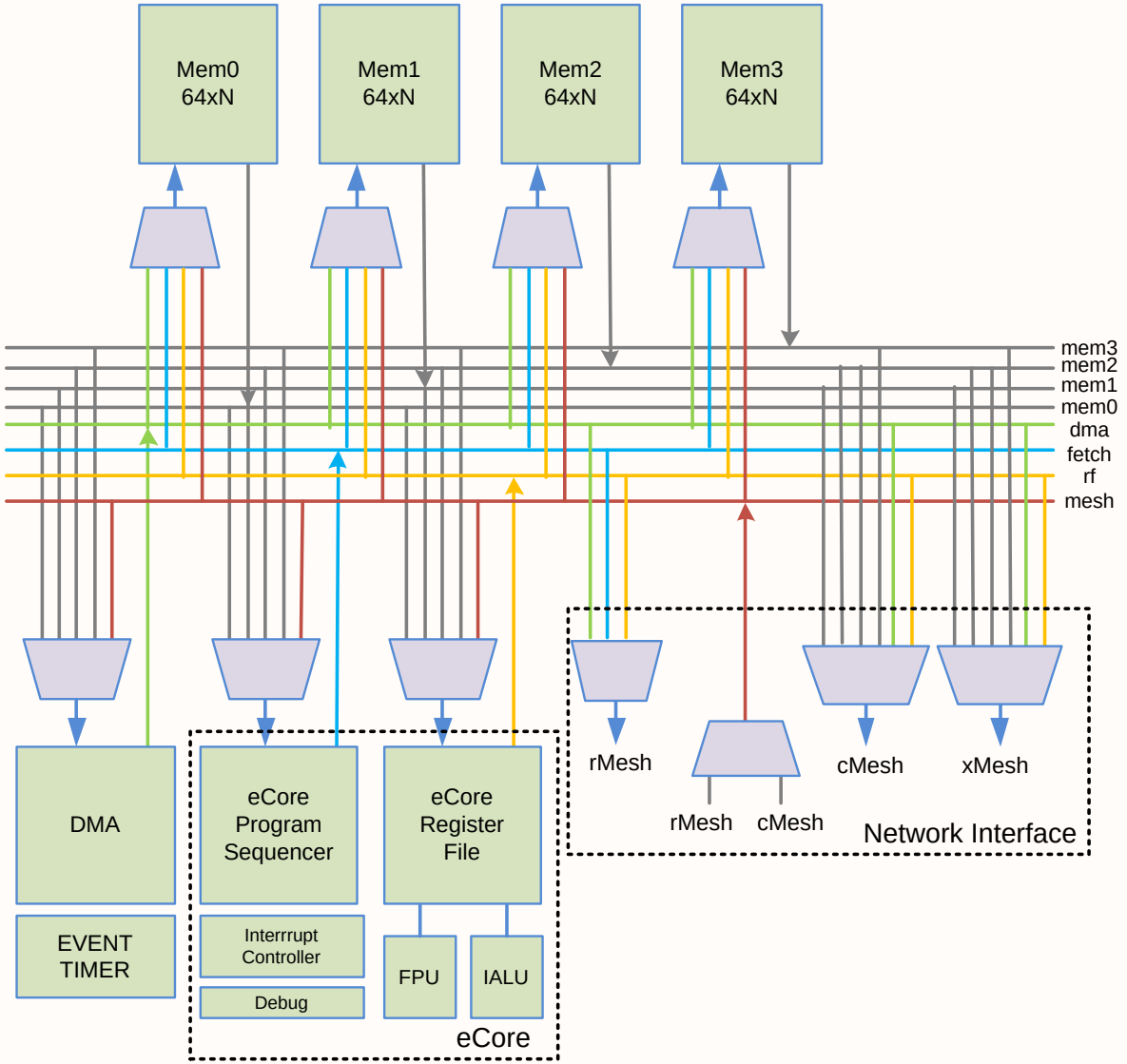
eCore CPU Pipeline

- Dual in-order issue
- Only integer/load/store issued in parallel with FPU
- Branch assumed taken
- 3 cycle penalty on missed branch
- 1 cycle integer pipeline
- 2 cycle load pipeline
- 4cycle floating point pipeline
- Fully interlocked



eCore Micro-Architecture

- Four memory banks
- 64bit * 4 access per clock cycle
- Fixed priority arbiter at every bank
  - External (highest)
  - Load-store
  - Instruction-fetch
  - DMA
- All resources memory mapped

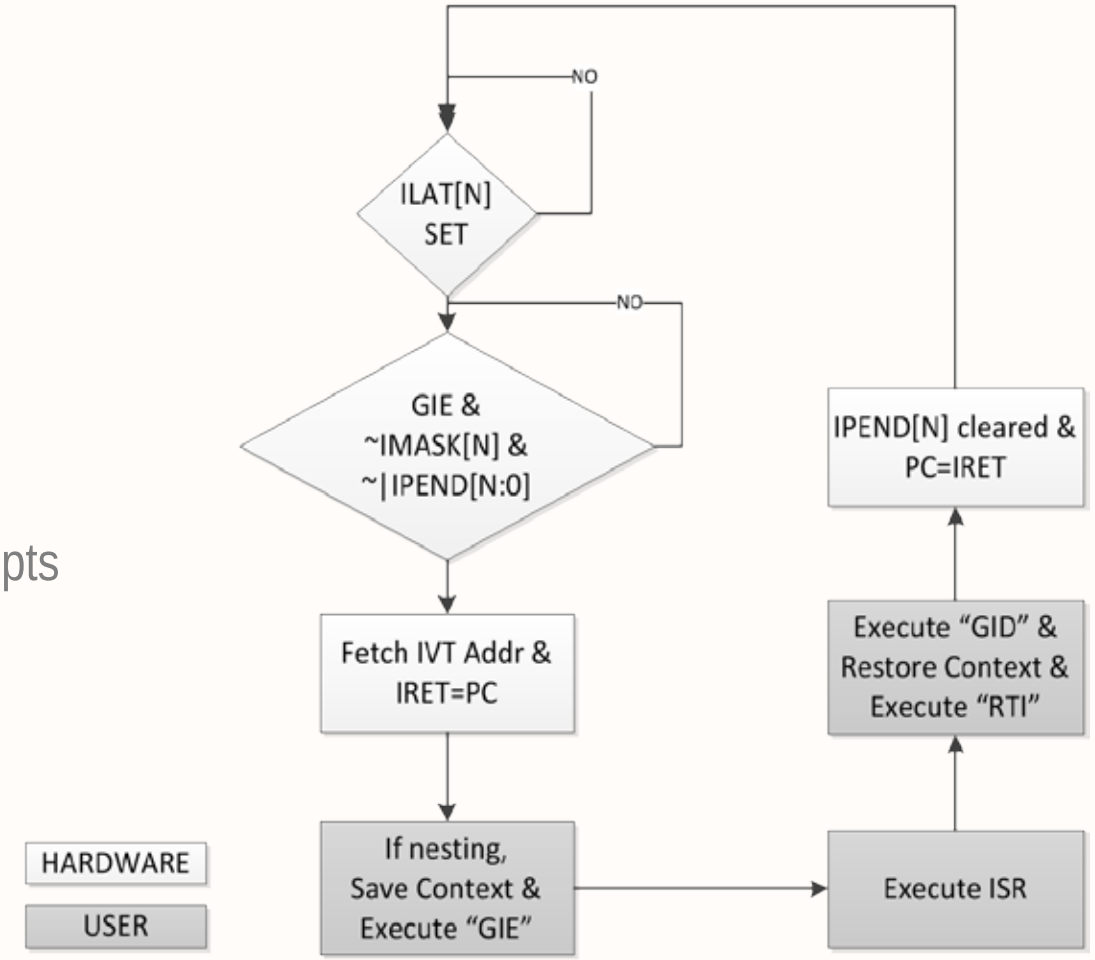


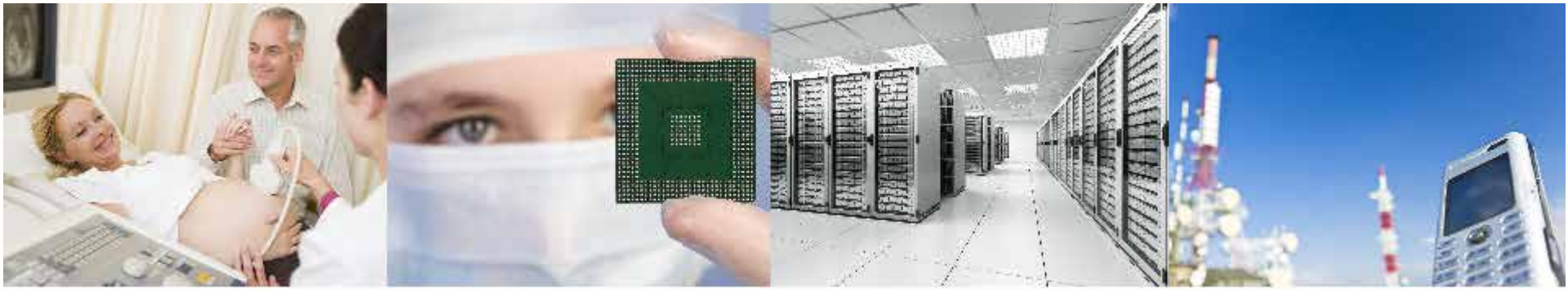
DMA Features

- Two DMA Channels per Processor Node
- 8GB/sec Per Processor Transfer Bandwidth
- Unrestricted memory to memory bulk transfers
- Flexible DMA striding
- 1D/2D Operations
- Programmable DMA Descriptors
- Full Chaining Support
- Traffic Throttle
- Interrupt generation

Interrupt Controller

- Interrupts:
  - Reset (highest)
  - SW Exception
  - Memory fault
  - Timer0
  - Timer1
  - DMA0
  - DMA1 (lowest)
- Features:
  - Full nested priority interrupts
  - Fast/slow ISR methods

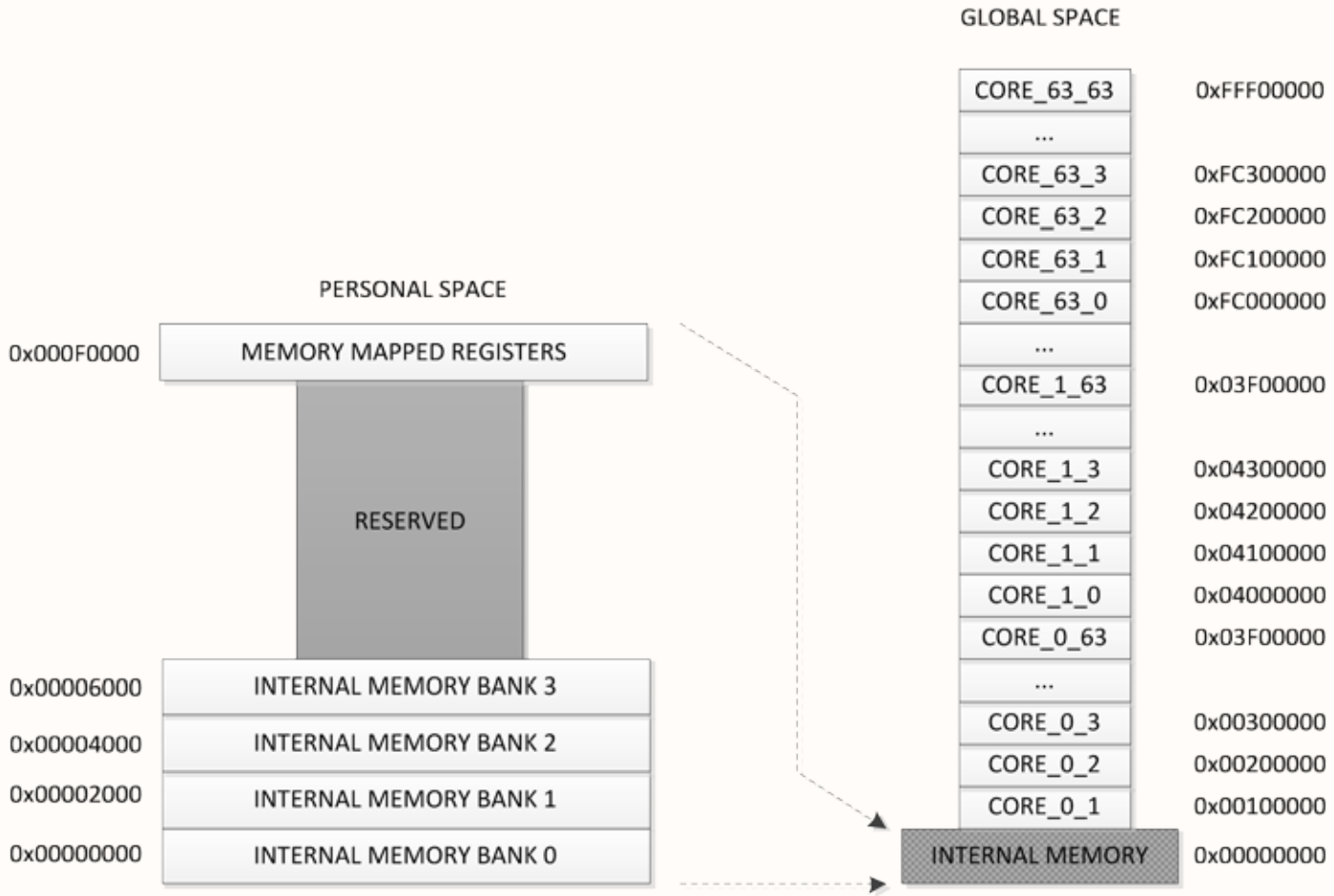




Epiphany Network

adapteva

Memory Architecture

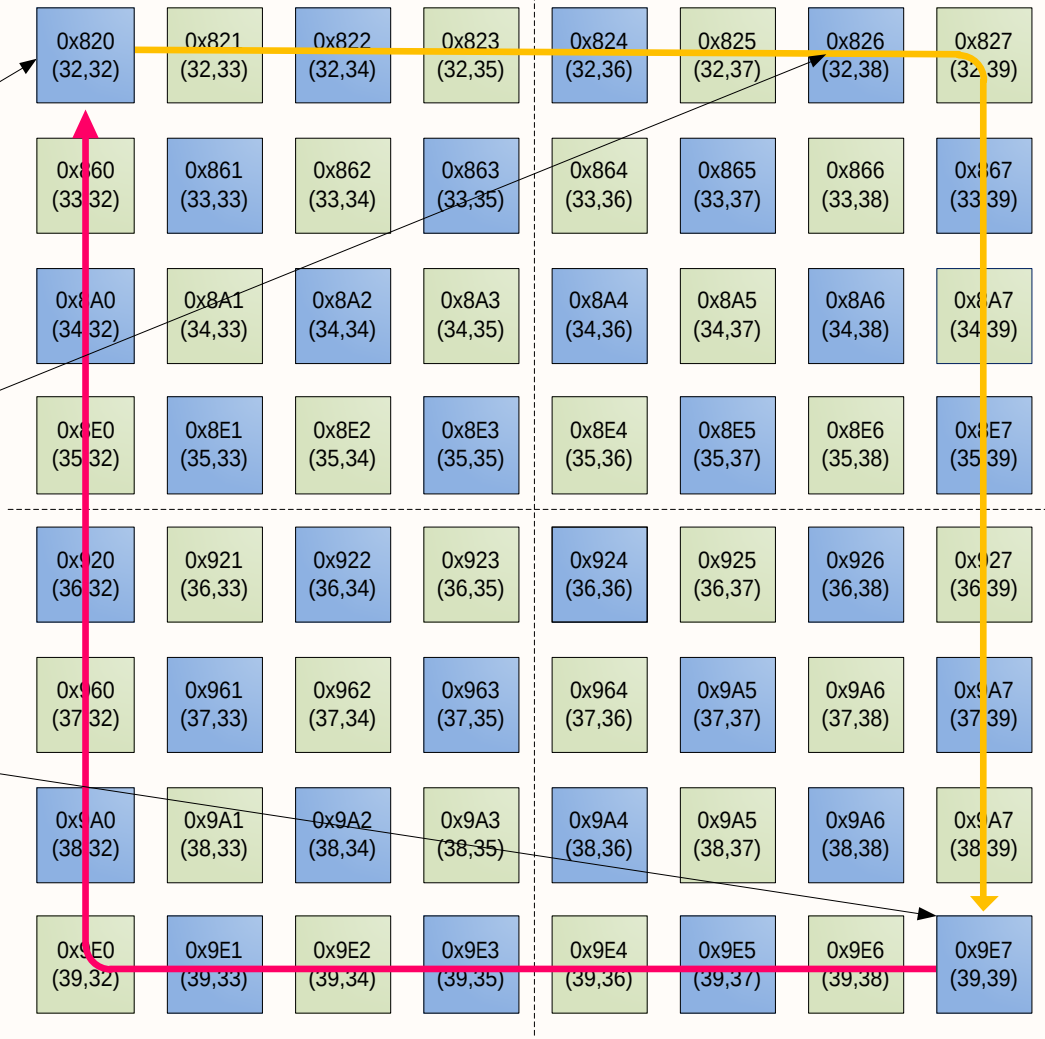


Memory/Routing Architecture

1. Core 32,32 sends a read request to core 39,39 and specifies its own core id(39,39) as a return (source) address

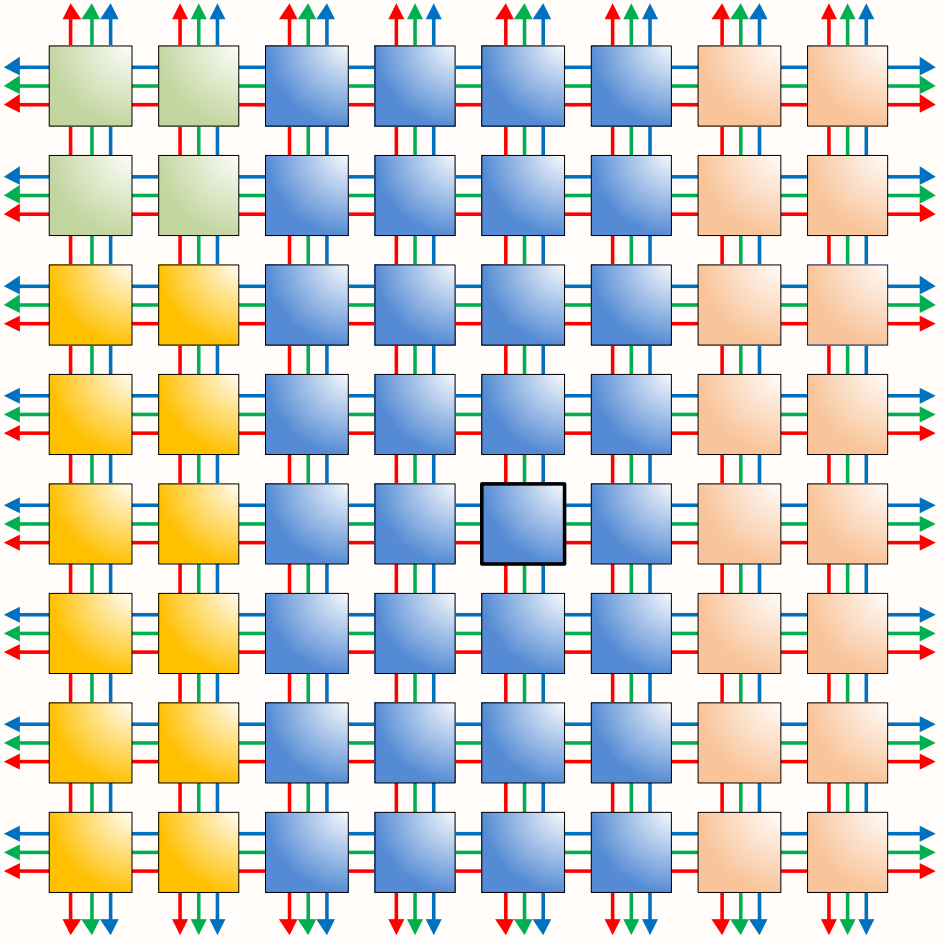
2.Along the path, routing nodes compare the transaction address to the routers hardcoded ID and decide if the transaction should be sent north, east, south, or west.

3. Core 39,39 receives read transaction, fetches data from memory and returns the data to core 32,32 (source) using the write network.



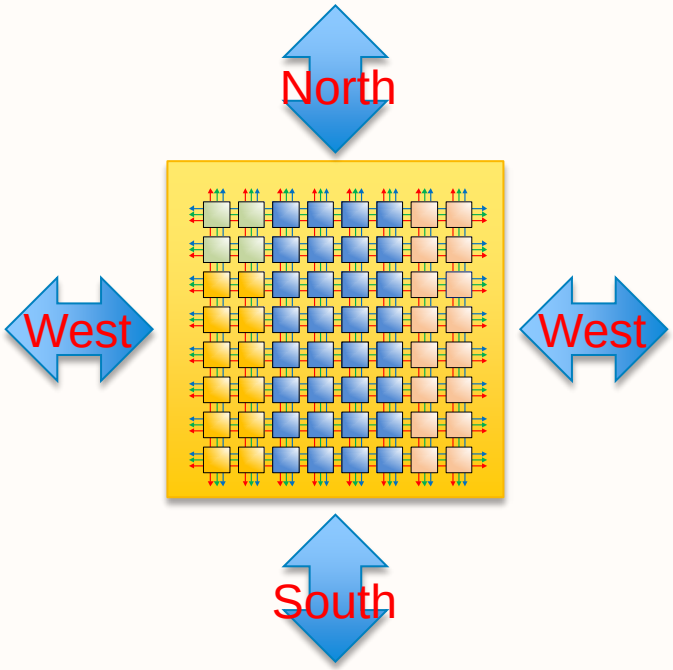
Scalable Low Power Network-On-Chip

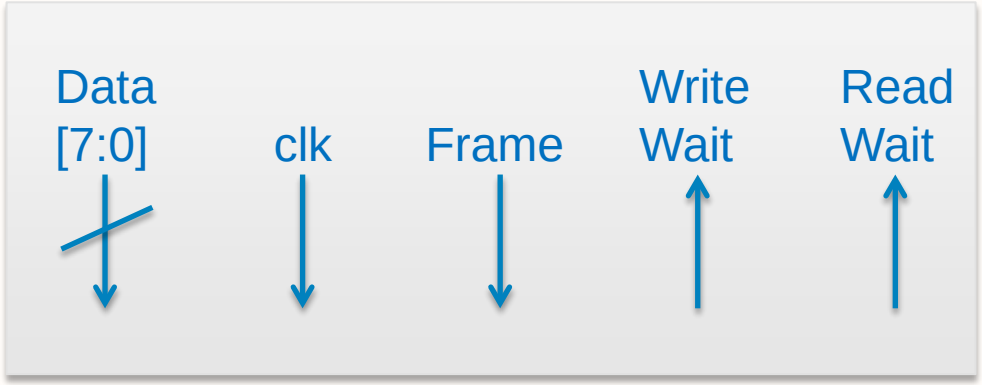
- Features:
  - Completely memory mapped
  - Split read/write networks
  - Supports accelerators
  - Scalable to 1000's of agents
  - Atomic 8 byte packets
- Performance (64 cores):
  - 1 GHz operation
  - 64 Billion Messages/sec
  - 1.5ns delay/hop
  - 0.8 TB/sec on chip BW
  - 30ns on round trip delay



Extended Off-chip I/O Protocol

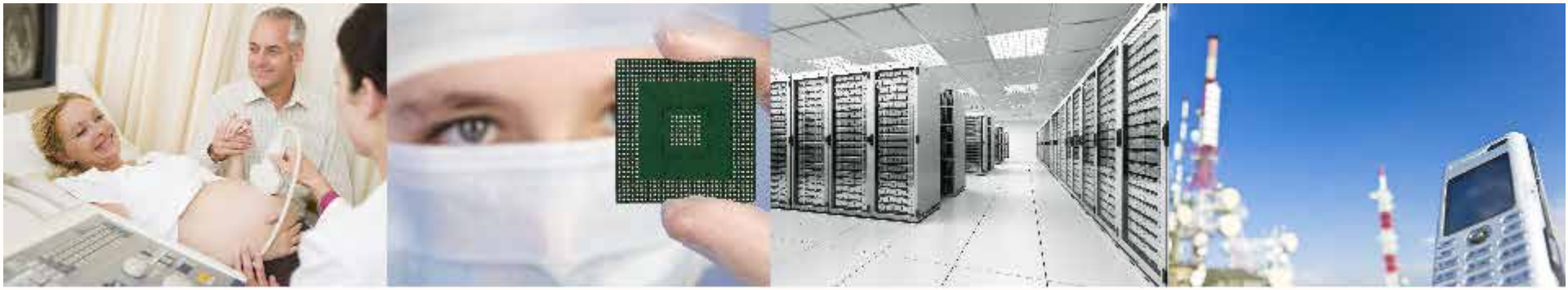
- North, East, West, South
- 4:1 NOC Link Muxing
- Memory mapped protocol
- Full push-back support
- Automatic streaming
- Enables glue-less chip scaling
- Low overhead
- Source Synchronous Clocking
- Designed to interface with low cost FPGAs
- 1 GB/sec off-chip BW per link





Advanced Multicore Features

- Atomic Test-and-Set at any local memory location
- Hardware synchronization
- Active messages
- Multicast data transfer
- Memory and Communication Protection
- Multicore breakpoints



Epiphany Based Products

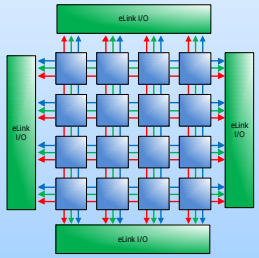
adapteva

Adapteva IP Offering

	E1G3	E16G3	E1G4	E16G4	E64G4	E256G4	E1KG4	E4KG4
Cores	1	16	1	16	64	256	1024	4096
Process Geometry	65G	65G	28LP	28LP	28LP	28LP	28LP	28LP
Max Frequency (MHz)	1000	1000	700	700	700	700	700	700
Performance (GFLOPS/sec)	2	32	1.4	22	88	352	1408	5632
Performance (CoreMark)	1288	1288*16	900	900*16	900*64	900*256	900*1k	900*4k
Peak Energy Efficiency (GFLOPS/W)	35	35	70	70	70	70	70	70
Total Area (mm^2)	0.5	8.96	0.13	2.05	8.2	32.7	131.1	524.3

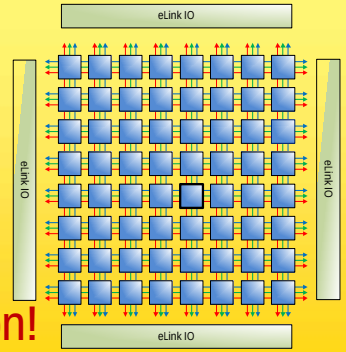
Technology Status

- 16-core Microprocessor
- 32 GFLOPS
- 35 GFLOPS/W
- 65nm technology
- Sampling since May 2011



Multicore Benchtop Evaluation Kits

- 64-core Microprocessor
- 100 GFLOPS
- 70 GFLOPS/W
- 28nm technology
- Out of fab, sampling soon!



COTS DSP boards from BittWare

Adapteva Product Potential

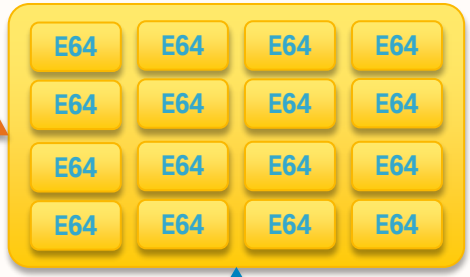
COTS Boards

VME, VPX, FMC AMC, PCIe, XMC

System On Chip

ANSI-C + Floating Point

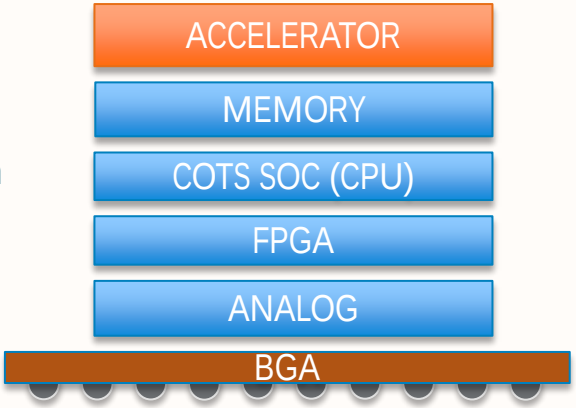
FPGA Coprocessor

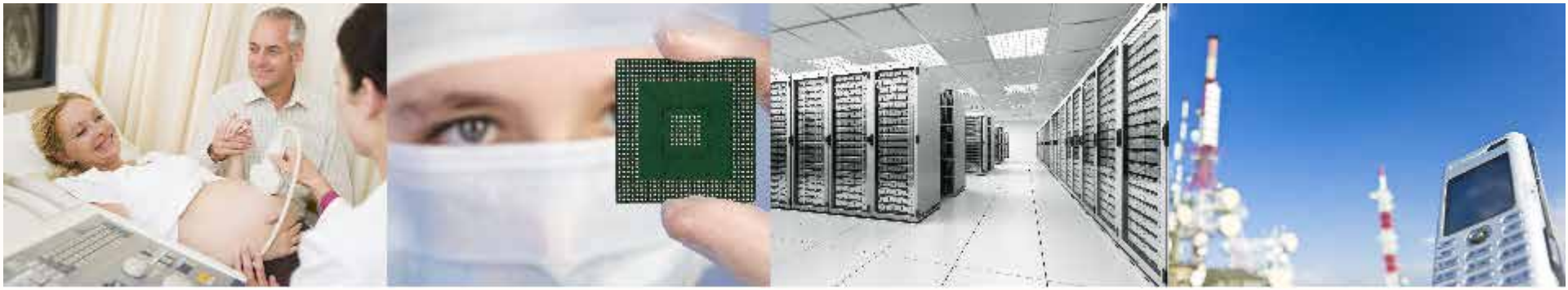


Sensor → FPGA → Analytics (Microprocessor)

System In Package

+Low Power
+Right Interfaces
+TSV
+Distributed Arch





Competitive Survey

adapteva

Epiphany vs ARM

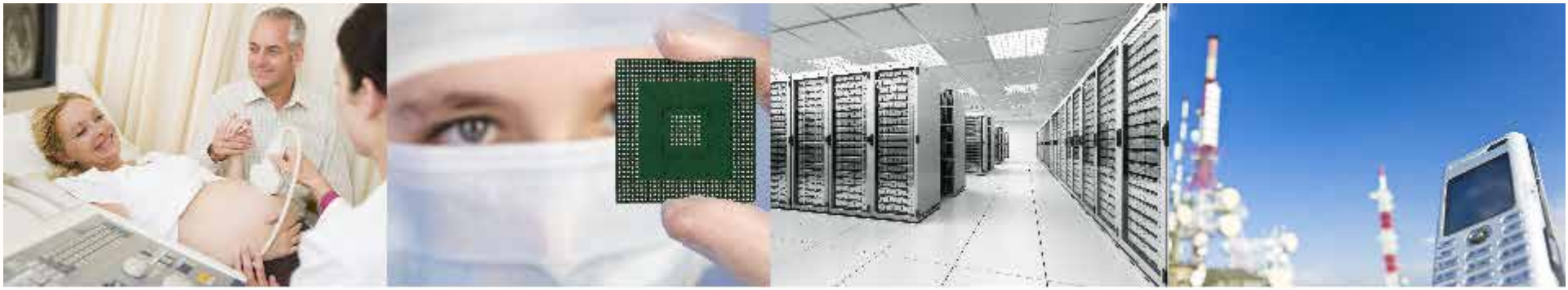
	Epiphany-16	A9 Dual Core	Note
Process	28LP	40G	Based on shipping silicon
Cores	64	2	ARM doesn't scale
Max Frequency	0.8GHz	2GHz	Custom design will close gap
Power Consumption	1.2	1.9	Individual core power-down
SIMD	No	Yes	SIMD doesn't scale
Performance (GFLOPS)	100	12.8	SIMD makes it much harder tor reach peak performance
CoreMark Score	80,528	9,164	We run any code well!
ANSI-C programmable	Yes	Yes	Enables developers

Adapteva offers an C-programmable API, giving all developers a platform that is as accessible as ARM, but with much better performance. We provide server level performance in a smartphone.

Epiphany vs GPGPU

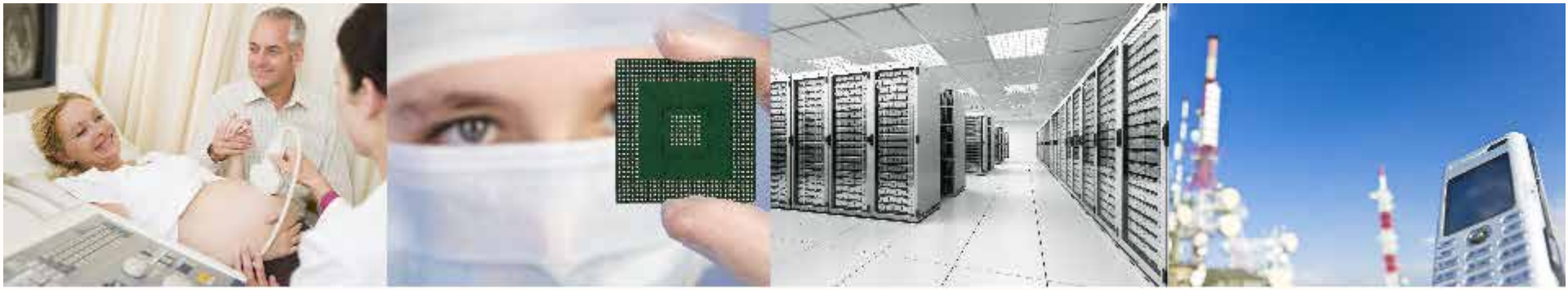
	Epiphany-16	Mali-400MP	Note
Process	28LP	65G	Our numbers are based on shipping silicon products
Cores	64	1	GPGPU's don't scale
Max Frequency	0.8GHz	395MHz	
GFLOPS/mm^2	12.5	1	
Power Consumption	1.2W	n/a	Related to area efficiency
SIMD	No	Yes	SIMD doesn't scale
Performance (GFLOPS)	100	~4	SIMD makes it much harder to reach peak performance
CoreMark Score	80,528	0	Mali is not C-programmable
ANSI-C programmable	Yes	No	

GPGPUs have good performance but are difficult to use and would never get widespread adoption among developers. openCL is not the answer!



Break…

adapteva



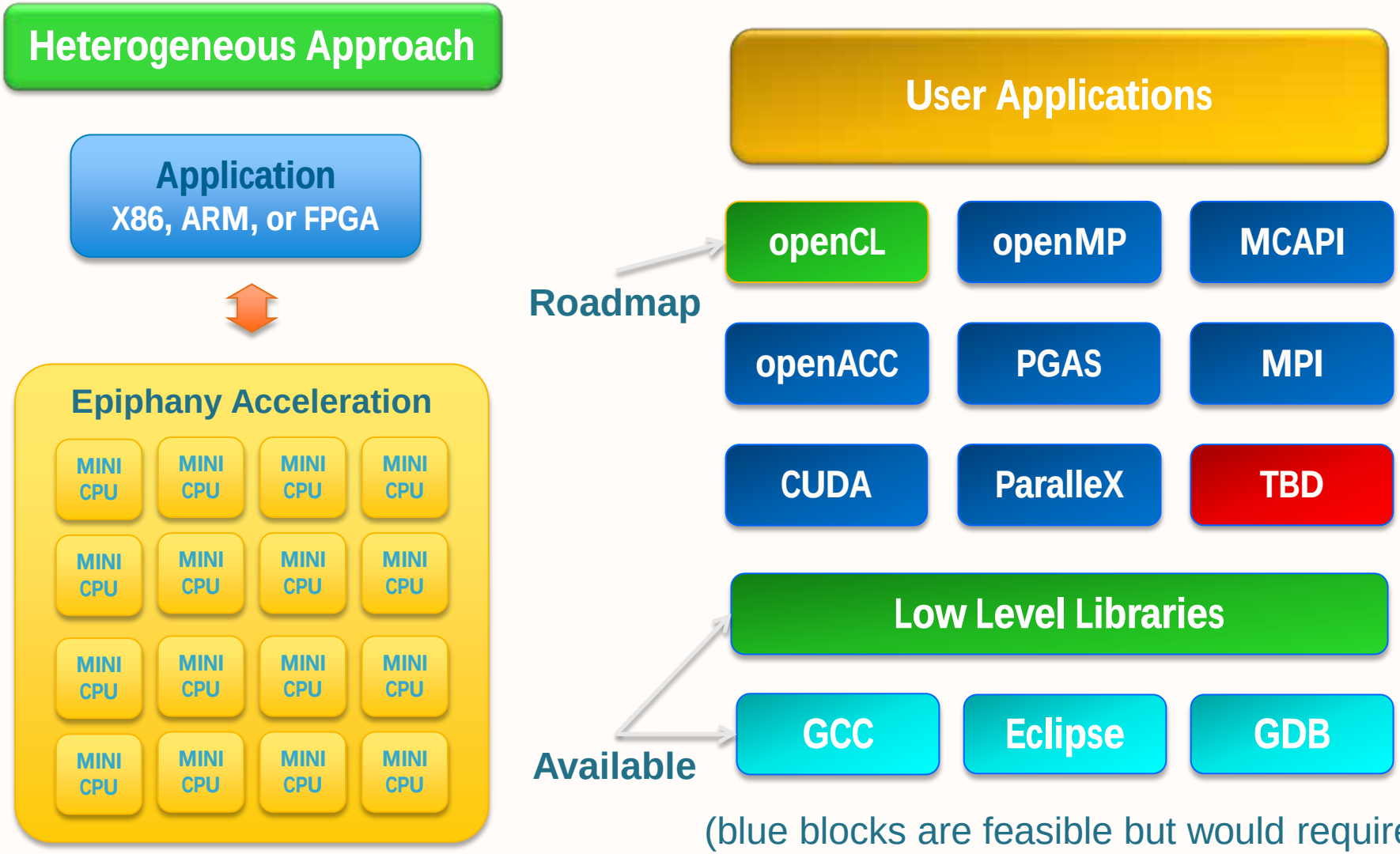
Programming Model

adapteva

A Programmer's Platform

- Really, truly ANSI-C programmable:
  - Each processor is a standalone microprocessor
  - 50-80% of peak performance from straight C-code
  - No SIMD, no VLIW, no pragmas, no proprietary libraries!!
  - Can be effectively used by anyone with an introductory course in C-programming
- Great for high performance:
  - Architecture transparency
  - Shared memory architecture
  - Orthogonal instruction set, large register file, shallow pipeline
  - No caches to get in the way or real time hard deadlines

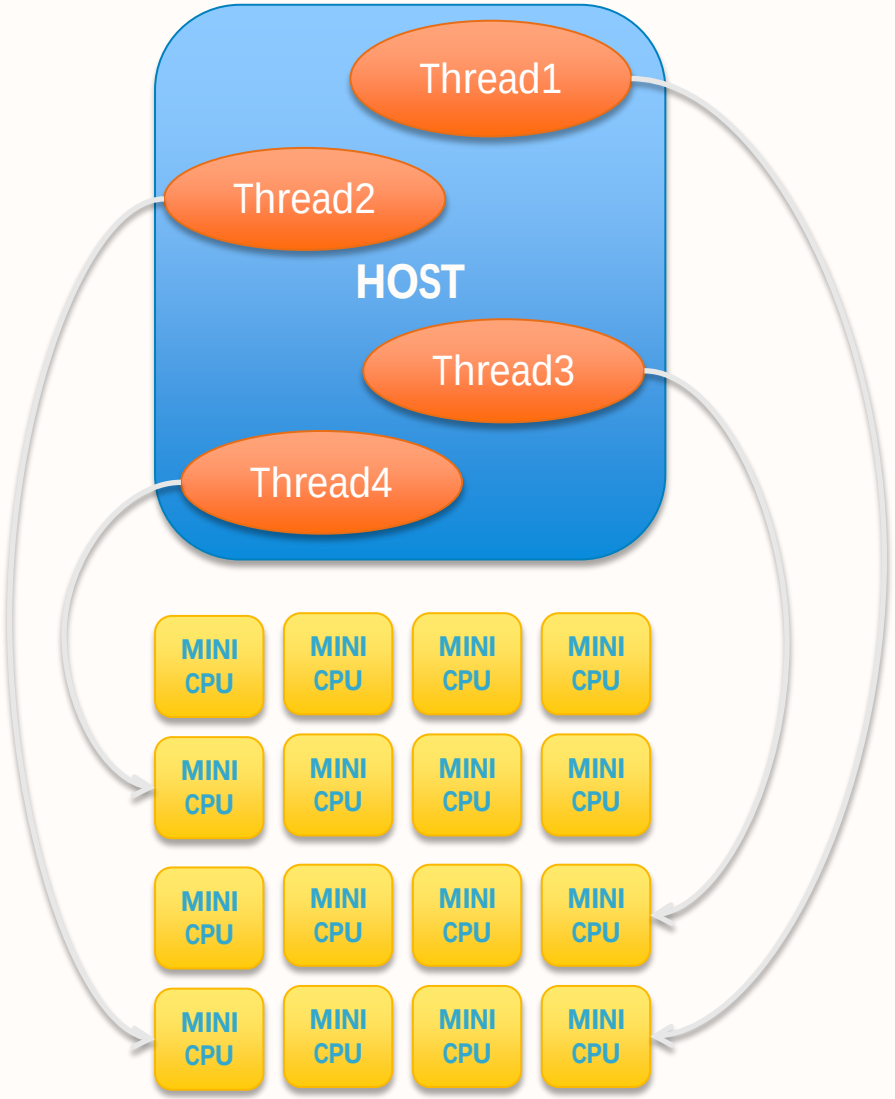
Epiphany Programming Model



(blue blocks are feasible but would require partner investment)

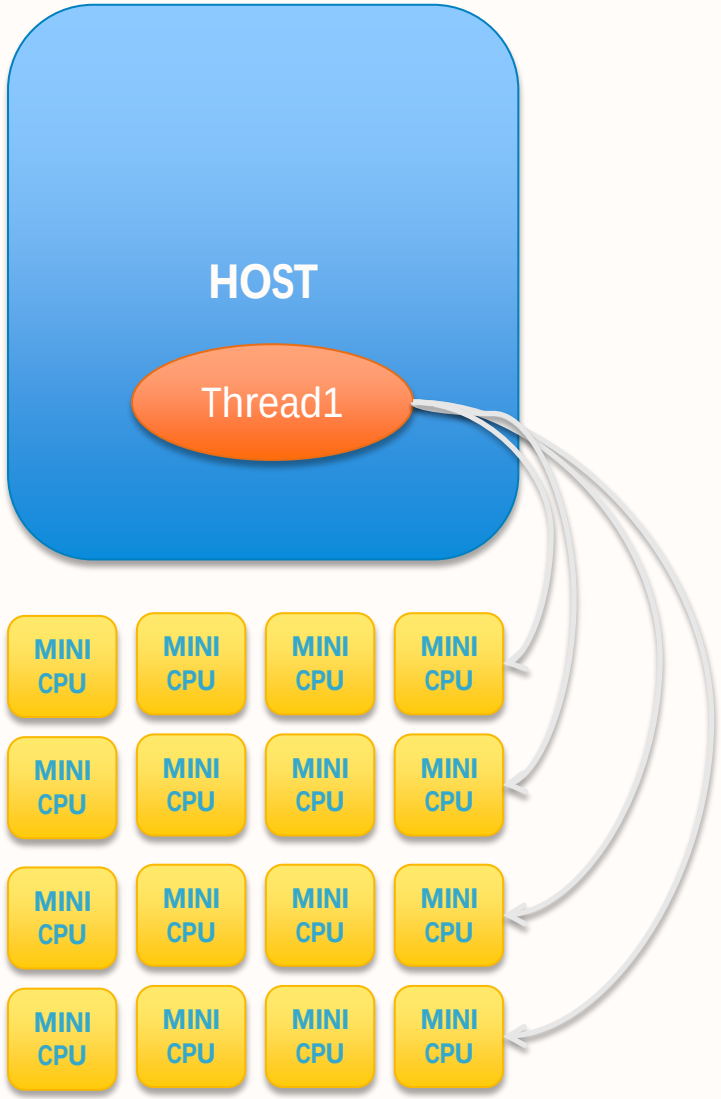
Use Case #1: Worker Bees

- Advantages:
  - Threads gets a 2GFLOPS boost
  - Very chunky parallelism
  - No need for “parallel programming”
- Disadvantages:
  - Limited latency improvement
  - Bandwidth bottleneck?

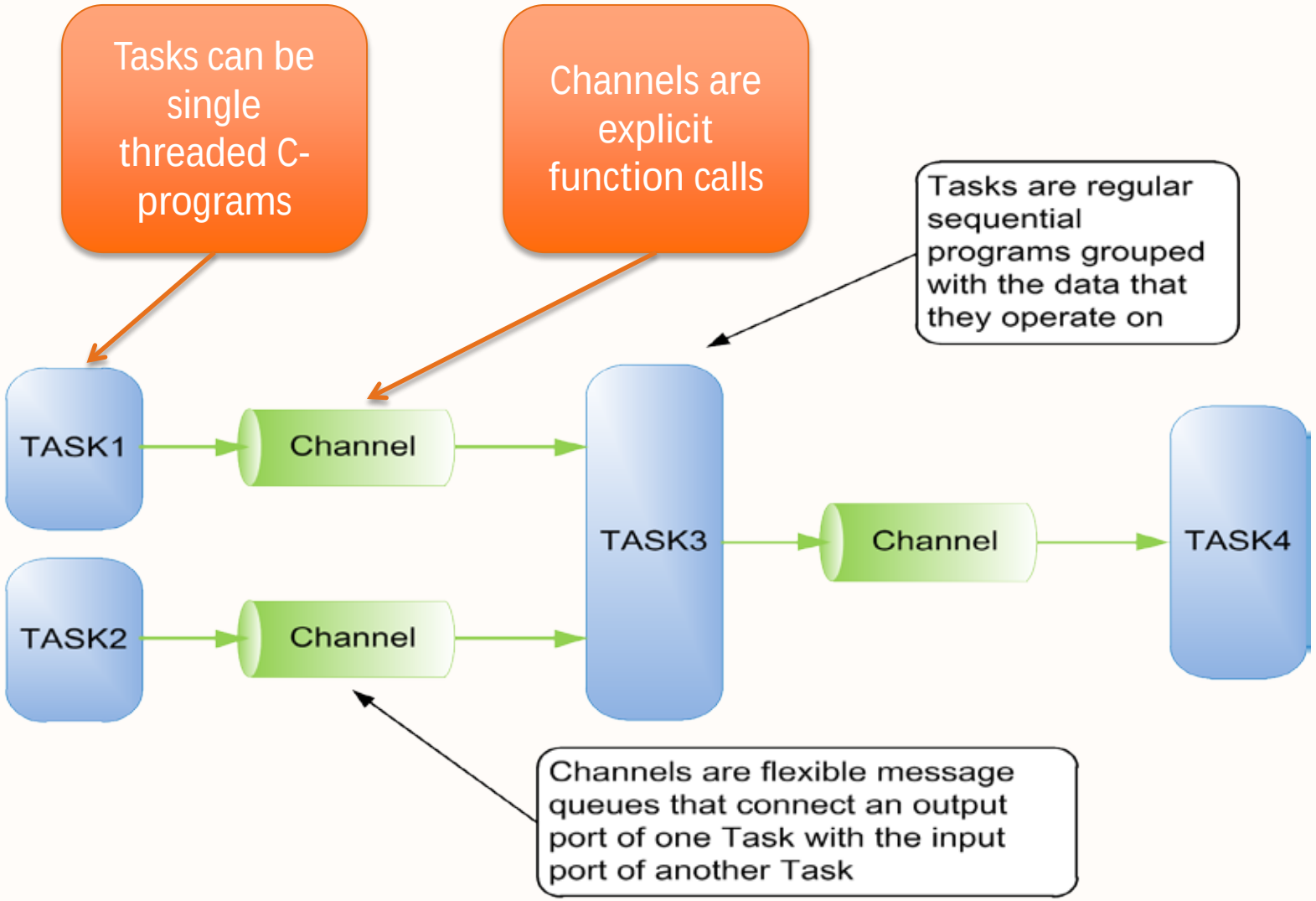


Use Case #2: Parallel Coprocessor

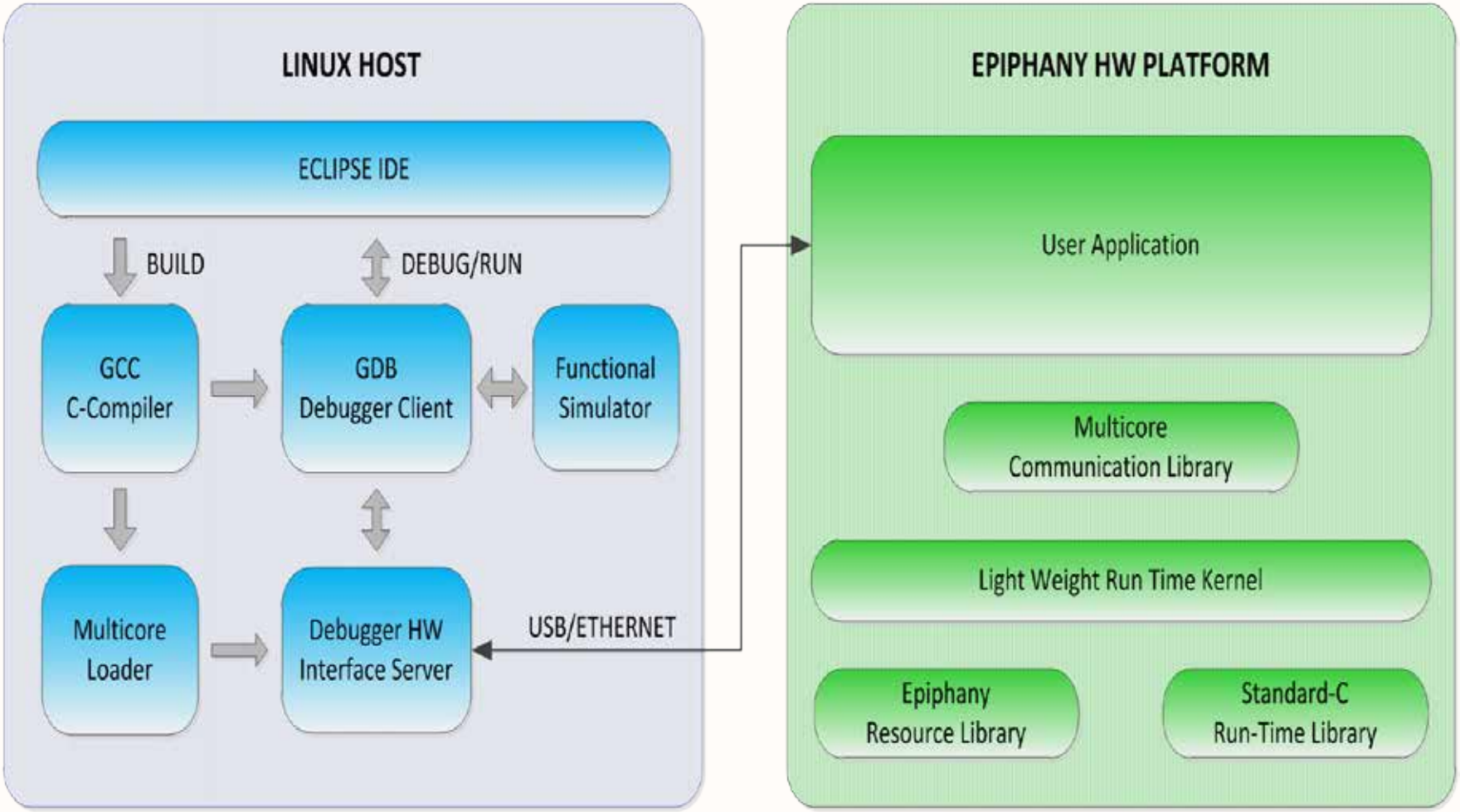
- Advantages:
  - Use one master thread to map to multiple processors
  - Leverages openCL
  - High throughput
- Disadvantages:
  - Coprocessor bottlenecks



Use Case #3: Streaming

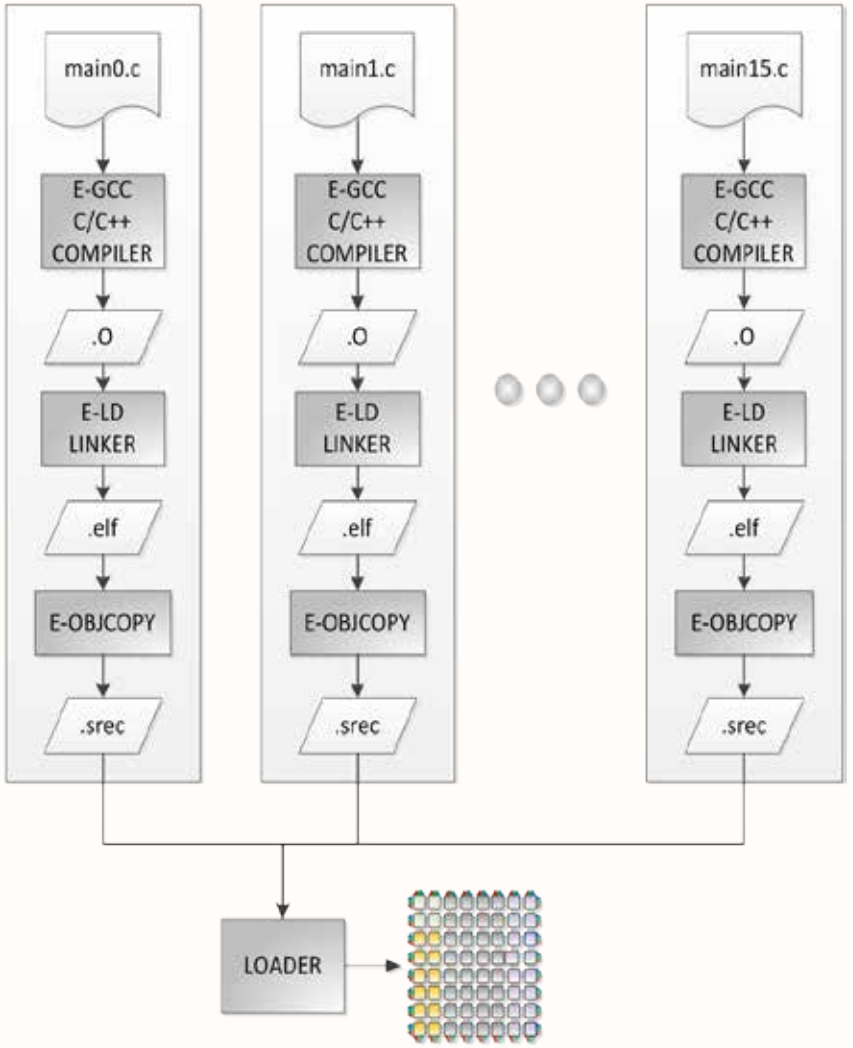


Epiphany SDK



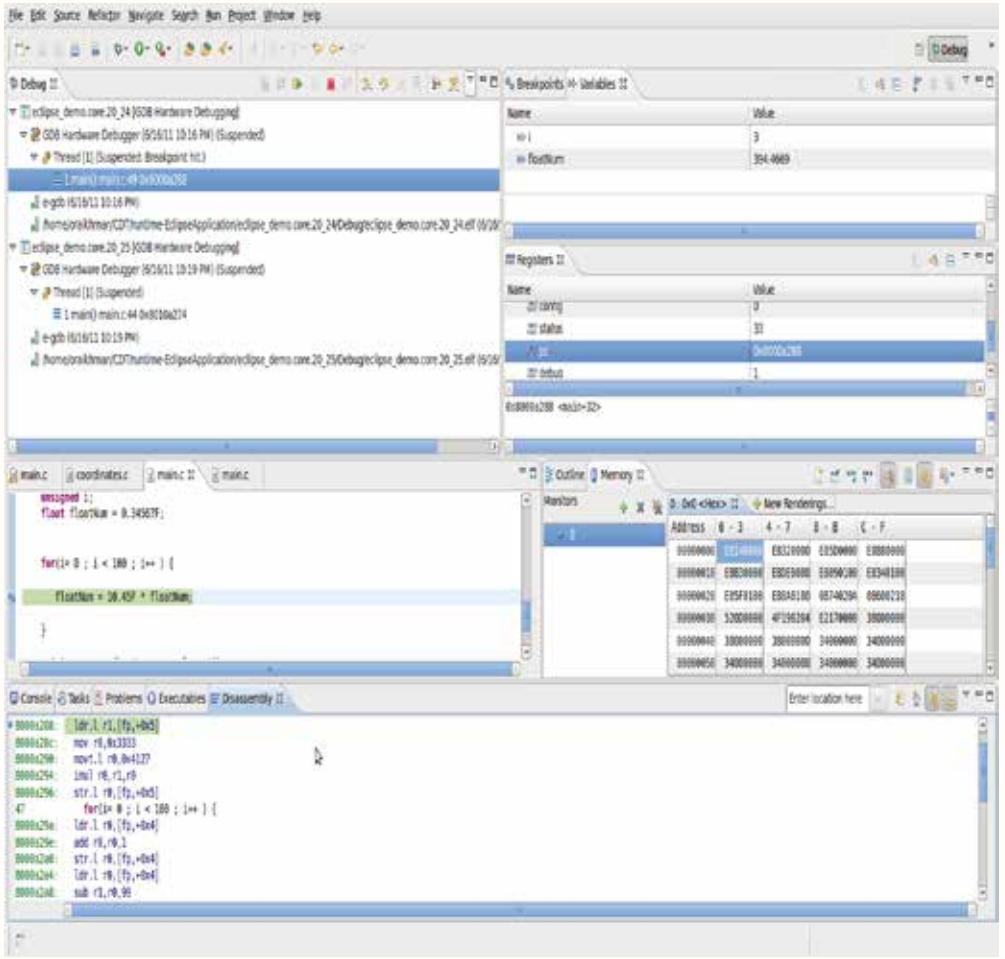
Compilation Flow

- Standard compile flow for each core
- One ELF file per core
- Symbols not shared between ELFs
- Currently static code only
- Working on relocatable code



Integrated Design Environment

- Based on Eclipse + CDT
- Multicore Project Management
- Multicore Debugger
- Multicore Loader
- Features:
  - Source level debugging
  - Direct hardware debugging
  - Program loading



Compiler

- Introduction:
  - Based on GCC4.7
  - A state of the art compiler implementation, proven by benchmarks!
  - Most common gcc switches supported
  - Uses custom linker scripts to address multicore paradigm
  - Optimized for inner loop processing…but does well on most code!
- Command Line:
  - e-gcc hello_world.c –o hello_world.elf

Debugger

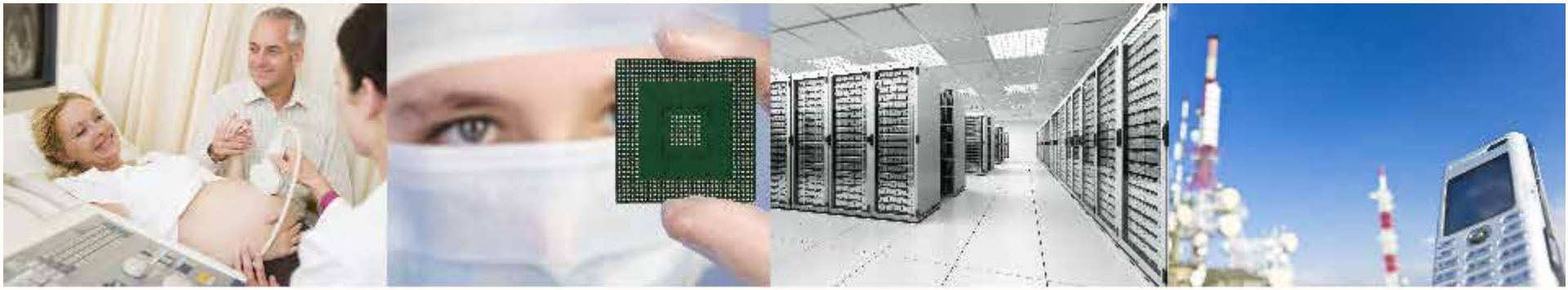
- Introduction:
  - Based on GDB 7.4
  - All the features you need in a debugger
  - Supports simulator and hardware debugging
  - Supports attaching itself to any core for debugging
- Command Line:
  - e-gdb hello_world.elf

Host Transaction Server

- Introduction:
  - Decouples application transaction from low level hardware communication
  - Connects GDB client with hardware target(s) through remote serial protocol
  - Loads program image onto hardware
  - Can be used to query hardware during normal operation
  - Runs once at board power-up
- Command Line:
  - e-server –xml emek3.xml

Epiphany Utility Library

- Function Groups:
  - Core-IDs
  - Timers
  - DMA
  - Mutex/Synchronization
  - Interrupts
  - System Register Access
- Goal:
  - Make sure users never need write assembly (unless they want to…)



Epiphany Benchmarks

adapteva

Single Core Floating Point Benchmarks

	Naïve C	Optimal C (cycles)	Theoretical	C-Efficiency
8x8 Matrix Multiplication	2852	773	512	66%
16 Tap FIR Filter (32 points)	1562	620	512	82%
Bi-quad IRR Filter(32 points)	n/a	1,047	768	73%
256 Point Dot-product	800	557	256	49%

1 day per C-benchmark

No Assembly Required

C Efficiency Key To Scaling Up User Cases

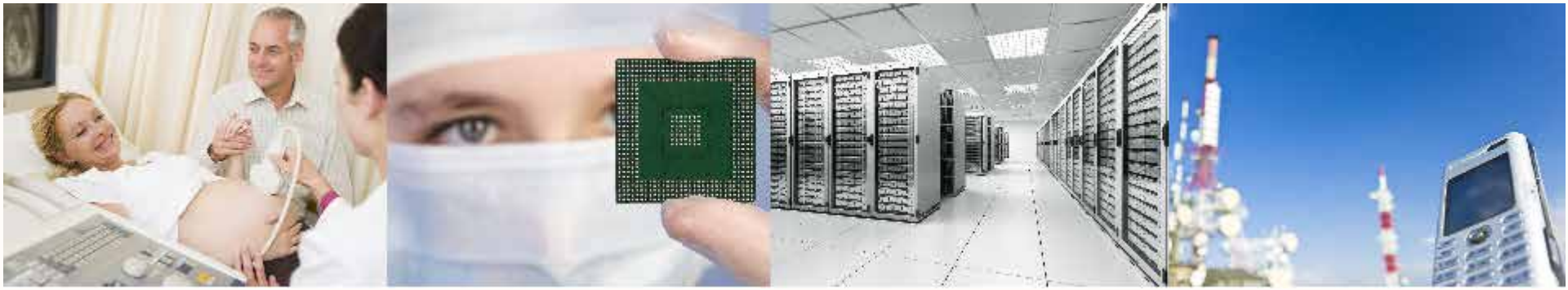
Multicore Integer Benchmark

CoreMark Intro

- ANSI-C benchmark
- No source code modification allowed
- Replacement for Dhrystone
- Integer benchmark (not our sweet-spot)
- www.coremark.org

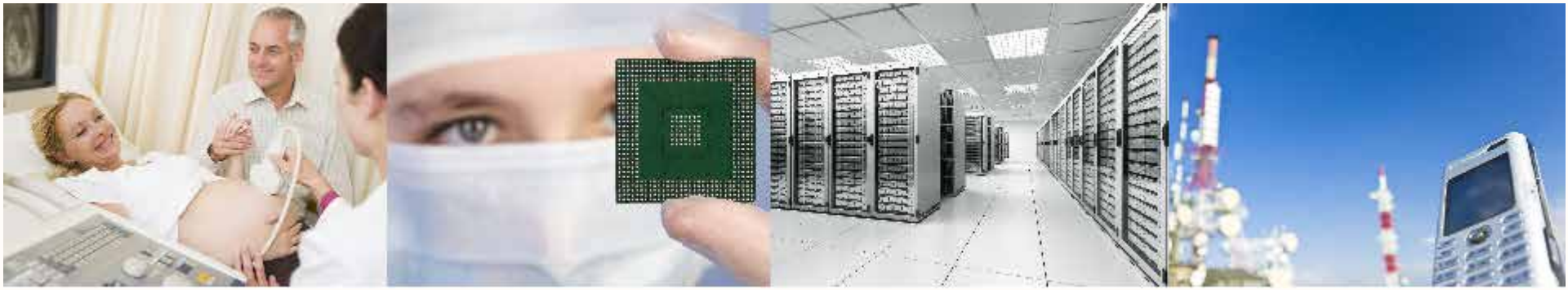
	Adapteva	Tilera	Intel	Nvidia
CoreMark Score	77,912	165,276	118,571	5,866
# Cores	64	35	8	2
Power	2	~50	~100	~2
1024-Core Roadmap Score	**2,493,184**	n/a	n/a	n/a

Note: Benchmark data from other companies are based on open information and is believed to be correct, but correctness cannot be guaranteed.



Break…

adapteva



Matrix Multiplication Demo

adapteva

Epiphany Parallel Programming

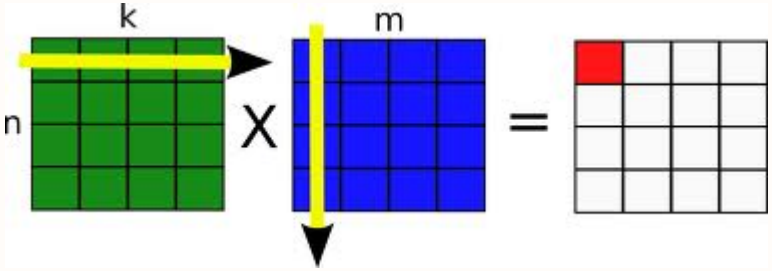
1. Define mathematical algorithm
2. Write single threaded C reference code
3. Define ways to parallelize mathematical algorithm
4. Map algorithm on hardware platform
5. Define hardware specific approach
6. Write code
7. Debug code

Epiphany Key Constraints

1. Geometrical Mapping (Core-id)
2. Chunky Parallelism (Reuse inner kernel)
3. Synchronization (Do it in software)
4. Communication (Do writes, not reads…)

1. Algorithm (Matrix Multiplication)

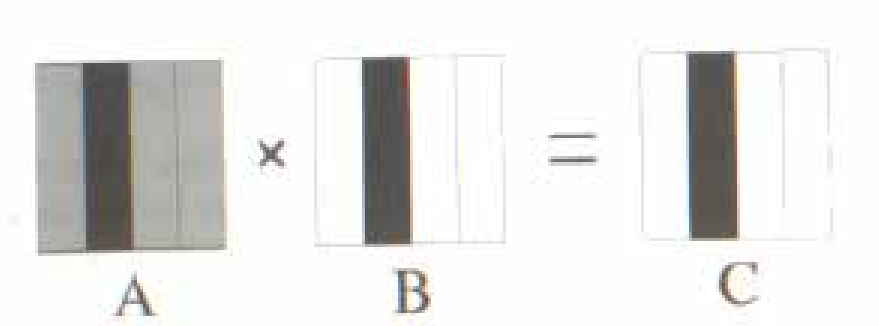
$$C_{i,j} = \sum_{i=1}^{k} A_{i,k} \cdot B_{k,j}$$

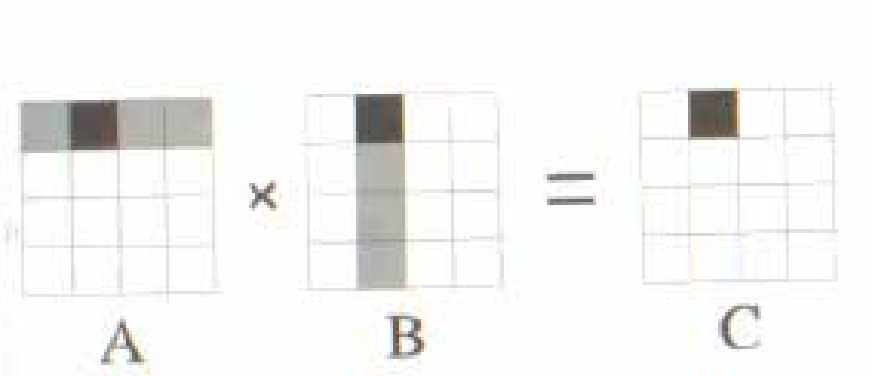


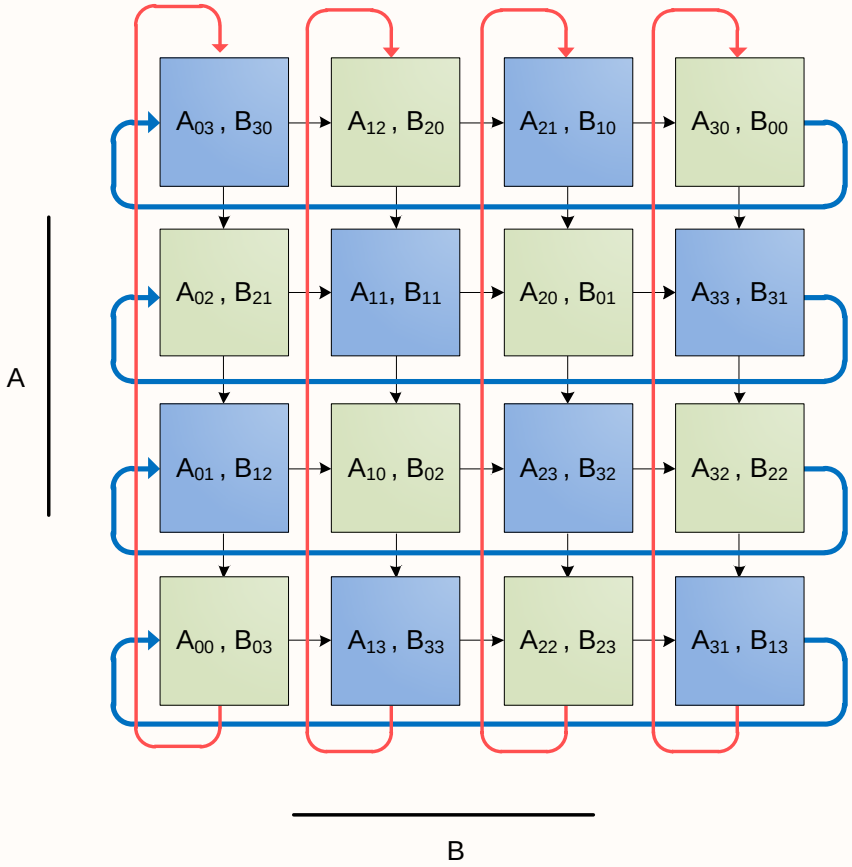
2. Matrix Multiplication Code

```
#pragma gss task input (A, B) inout (c)

static void block_addmultiply( double C[BS][BS], double A[BS][BS], double B[BS][BS]){
        int i, j, k_;
        for (i = 0; i < BS; i++)
                for (j = 0; j < BS; j++)
                        for (k =0; k < BS; k++)
                                C[i][j] += A[i][k] * B[k][j];
}

int main (int argc, char **argv){
        int i, j, k;
        initialize (argc, argv,  A, B, C);
        for (i = 0; i < N; i++)
                for (j = 0; j < N; j++)
                        for (k =0; k < N; k++)
                                block_addmultiply(C[i][j], A[i][k], B[k][j]);
        ...
}
```

3. Parallel Matrix Multiplication



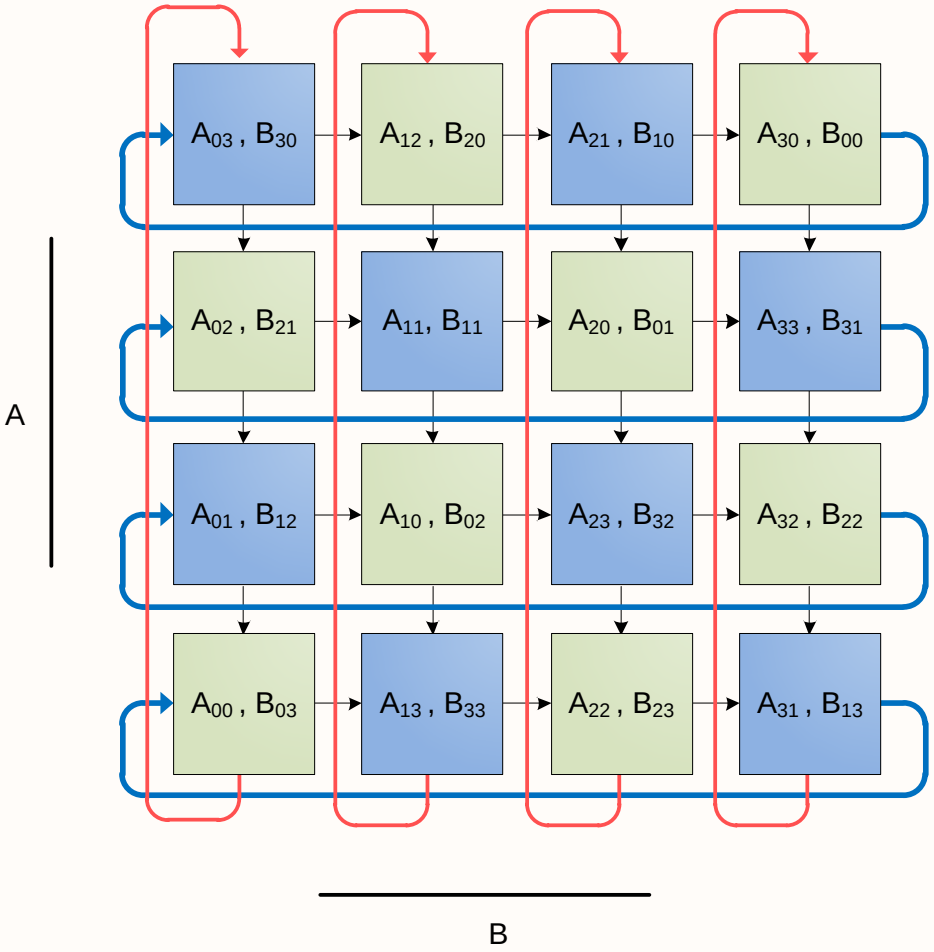




4. Epiphany Hardware Constraints

5. Chosen Algorithm

1. Copy data from host to SDRAM
2. Each core copies a 32x32 A,B blocks from SDRAM
3. Each core computes 32x32 multiple
4. Core shifts A/B matrix down, right
5. Complete step 3,4 four times
6. Store 128x128 C matrix in SDRAM
7. Go through step 2-6 N more times to compute complete C in SDRAM



6. Write code (first half)

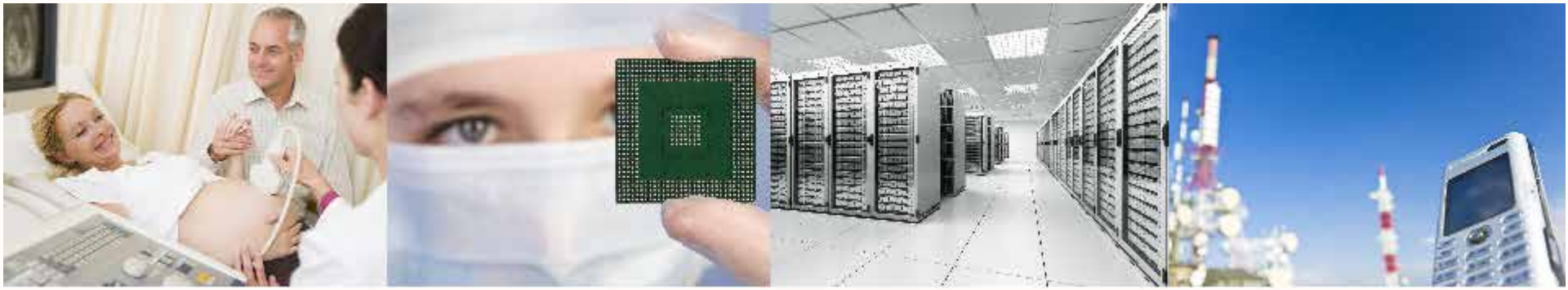
```
for (jm=0, jmp=0; jm<_Nchips; jm++, jmp+=_Schip){
//First clear the local result submatrix. The product result will be
matclr(me.bank[_BankC][_PING], _Score);
for (km=0, kmp=0; km<_Nchips; km++, kmp+=_Schip){
//Core loop through chip:
icp = me.row * _Score;
jcp = ((me.col + me.row) % _Nside) * _Score;
subcpy(Mailbox.A, _Smtx, (imp+icp), (kmp+jcp),
me.bank[_BankA][me.pingpong], _Score, 0, 0, _Score);
jcp = me.col * _Score;
icp = ((me.row + me.col) % _Nside) * _Score;
subcpy(Mailbox.B, _Smtx, (kmp+icp), (jmp+jcp),
me.bank[_BankB][me.pingpong], _Score, 0, 0, _Score);
//Multiply submatrices (inner product of row x column
for (kc=0; kc<_Nside; kc++){
//Core matmul:
matmac(me.bank[_BankA][me.pingpong], me.bank[_BankB][me.pingpong],
me.bank[_BankC][_PING], _Score);
```

6. Write code (second half)

```
//After multiplying the submatrices in each core, rotate A ,B
while (*me.tgt_synch == 1) {};
if (kc <= (_Nside - 1))
subcpy(me.bank[_BankA][me.pingpong], _Score, 0, 0,
me.tgt_bk[_BankA][me.pingpong], _Score, 0, 0, _Score);
*me.tgt_synch = 1;
//Rotate B banks vertically
while (*me.tgt_syncv == 1) {};
if (kc <= (_Nside - 1))
subcpy(me.bank[_BankB][me.pingpong], _Score, 0, 0,
me.tgt_bk[_BankB][me.pingpong], _Score, 0, 0, _Score);
*me.tgt_syncv = 1; me.pingpong = 1 - me.pingpong;
//Wait for peer sync
while ((me.synch == 0) || (me.syncv == 0)) {}; me.synch = me.syncv = 0; }}
//Extract the chip result to DRAM
icp = me.row * _Score; jcp = me.col * _Score;
subcpy(me.bank[_BankC][_PING], _Score, 0, 0, Mailbox.C, _Smtx, (imp+icp),
(jmp+jcp), _Score);
}return;}
```

6. Debug Code

- Finding asynchronous bugs is hard
- Best to avoid them, don't reinvent the wheel!!!
- Divide and conquer critical!
- If you have bugs:
  - GDB
  - Traces
  - Printf
  - Breakpoints



FIR Filter Demo

adapteva

65% Efficiency With C-code

```
for (rdp=0; rdp<_Ndata; rdp+=_Ntaps){
  for (wrp=0; wrp<_Ntaps; wrp+=4){
    dl[wrp+0] = dl[wrp+_Ntaps+0] = inp_data[rdp+wrp+0];
    dl[wrp+1] = dl[wrp+_Ntaps+1] = inp_data[rdp+wrp+1];
    dl[wrp+2] = dl[wrp+_Ntaps+2] = inp_data[rdp+wrp+2];
    dl[wrp+3] = dl[wrp+_Ntaps+3] = inp_data[rdp+wrp+3];
    fir[0] = fir[1] = fir[2] = fir[3] = 0;
    for (cp=0, dlp=(wrp+1); cp<_Ntaps; cp++, dlp++){
      fir[0] += c[cp] * dl[dlp + 0];
      fir[1] += c[cp] * dl[dlp + 1];
      fir[2] += c[cp] * dl[dlp + 2];
      fir[3] += c[cp] * dl[dlp + 3];
    }
  out_data[rdp+wrp+0] = fir[0];
  out_data[rdp+wrp+1] = fir[1];
  out_data[rdp+wrp+2] = fir[2];
  out_data[rdp+wrp+3] = fir[3];
}}
```