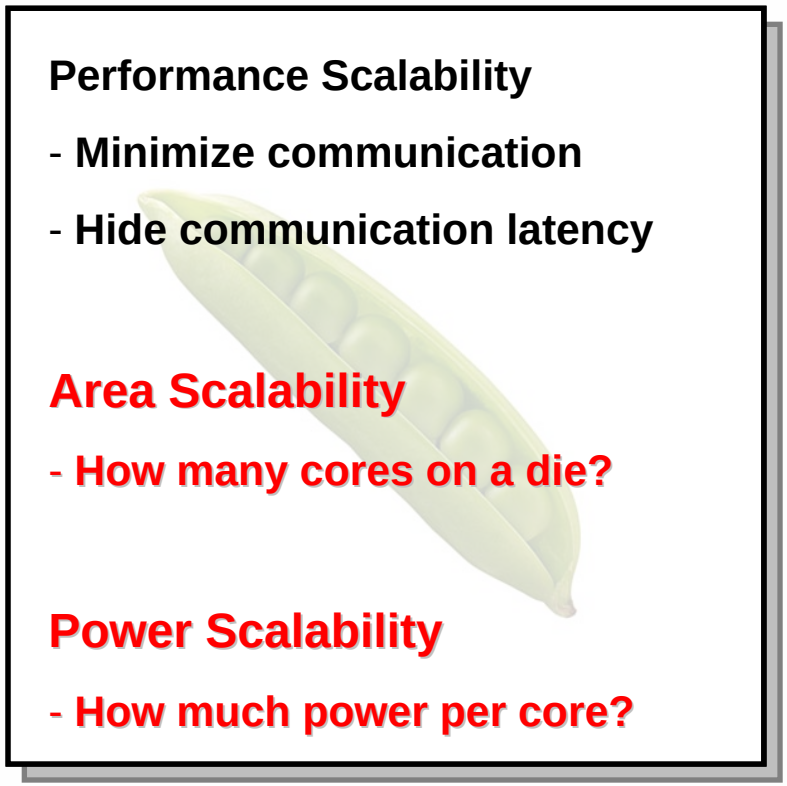
POD: A Parallel-On-Die Architecture

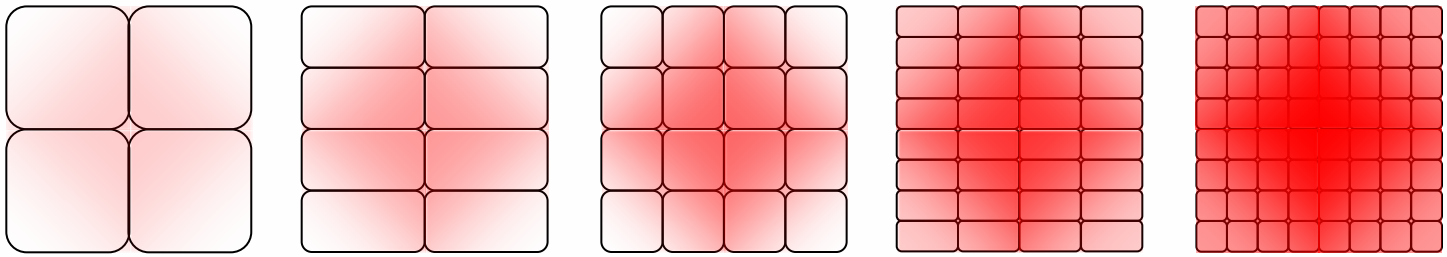
Dong Hyuk Woo	Georgia Tech
Joshua B. Fryman	Intel Corp.
Allan D. Knies	Intel Berkeley Research
Marsha Eng	Intel Corp.
Hsien-Hsin S. Lee	**Georgia Tech**

What So New with 'Many-Core'?

- **On Chip!**
  - **New definition of scalability!**



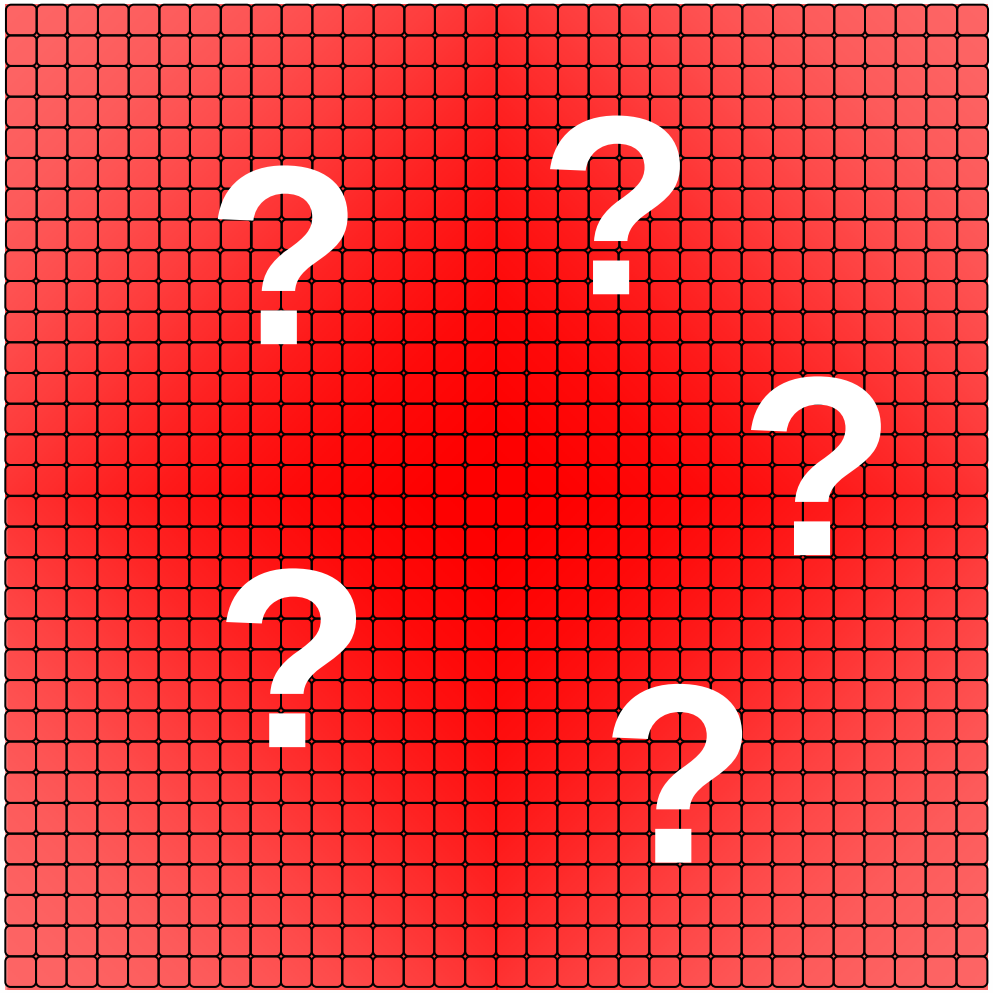
Scalable Power Consumption?



Technology (nm)	**65**	**45**	**32**	**22**	**16**
# of cores	**4**	**8**	**16**	**32**	**64**
Frequency (GHz)	**3.0**	**3.0**	**3.0**	**3.0**	**3.0**
Vdd (V)	**1.3**	**1.1**	**0.9**	**0.8**	**0.7**
Power / core (W)	**37.5**	**26.8**	**18.0**	**14.2**	**10.9**
Total power (W)	**150**	**215**	**288**	**454**	**696**

* The above pictures do not represent any current or future Intel products.

Thousand Cores, Power Consumption?



Some Known Facts on Interconnection

- **One main energy hog!**
  - **Alpha 21364:**
    - **20% (25W / 125W)**
  - **MIT RAW:**
    - **40% of individual tile power**
  - **UT TRIPS:**
    - **25%**

Interconnection Network related

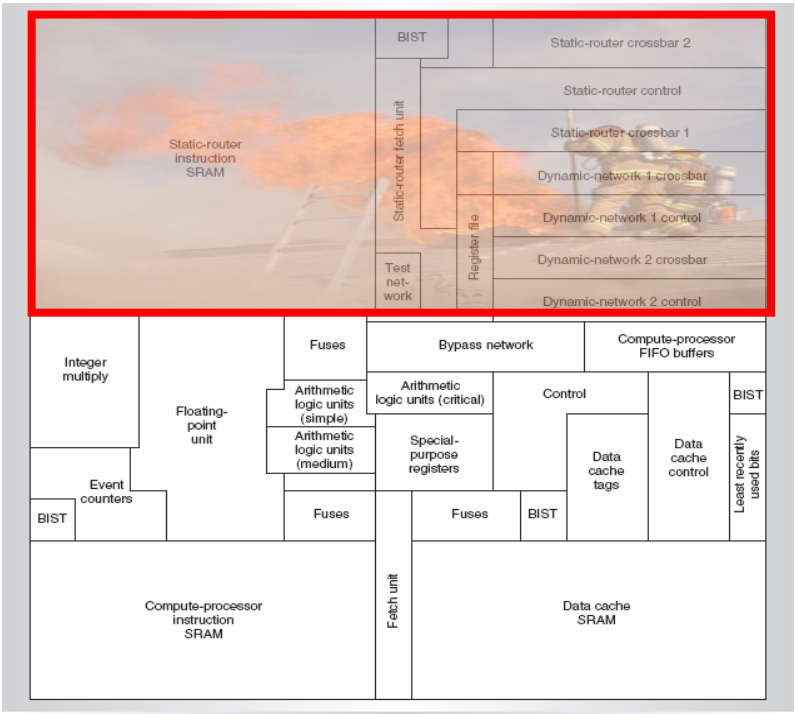


Figure 8. Raw tile floor plan.

- **Mesh-based MIMD many-core?**
  - **Unsustainable energy in its router logic**
  - **Unpredictable communication pattern**
    - **Unfriendly to low-power techniques**

POD: A Parallel-On-Die Architecture

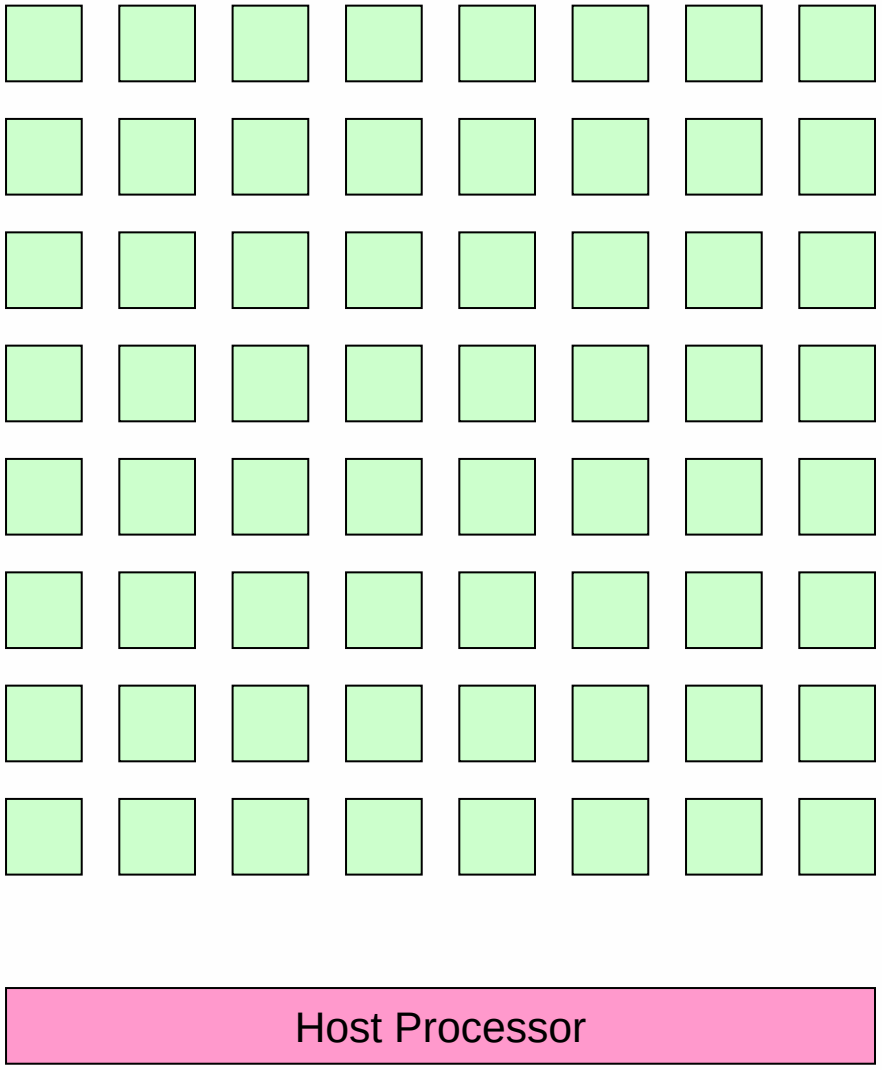
Design Goals

- **Broad-purpose acceleration**
  - **Scalable vector performance**
- **Energy and area efficiency**
- **Low latency inter-PE communication**
- **Best-in-class scalar performance**
- **Backward compatibility**

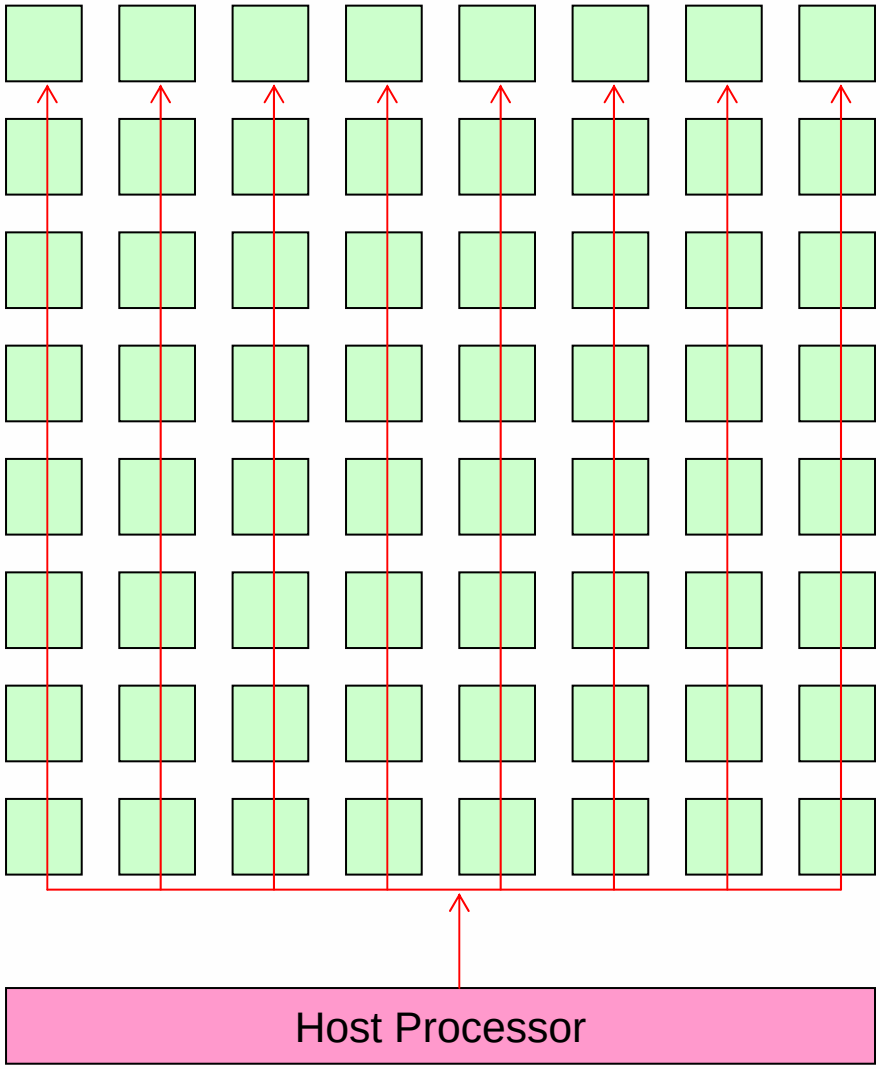
Host Processor

Host Processor

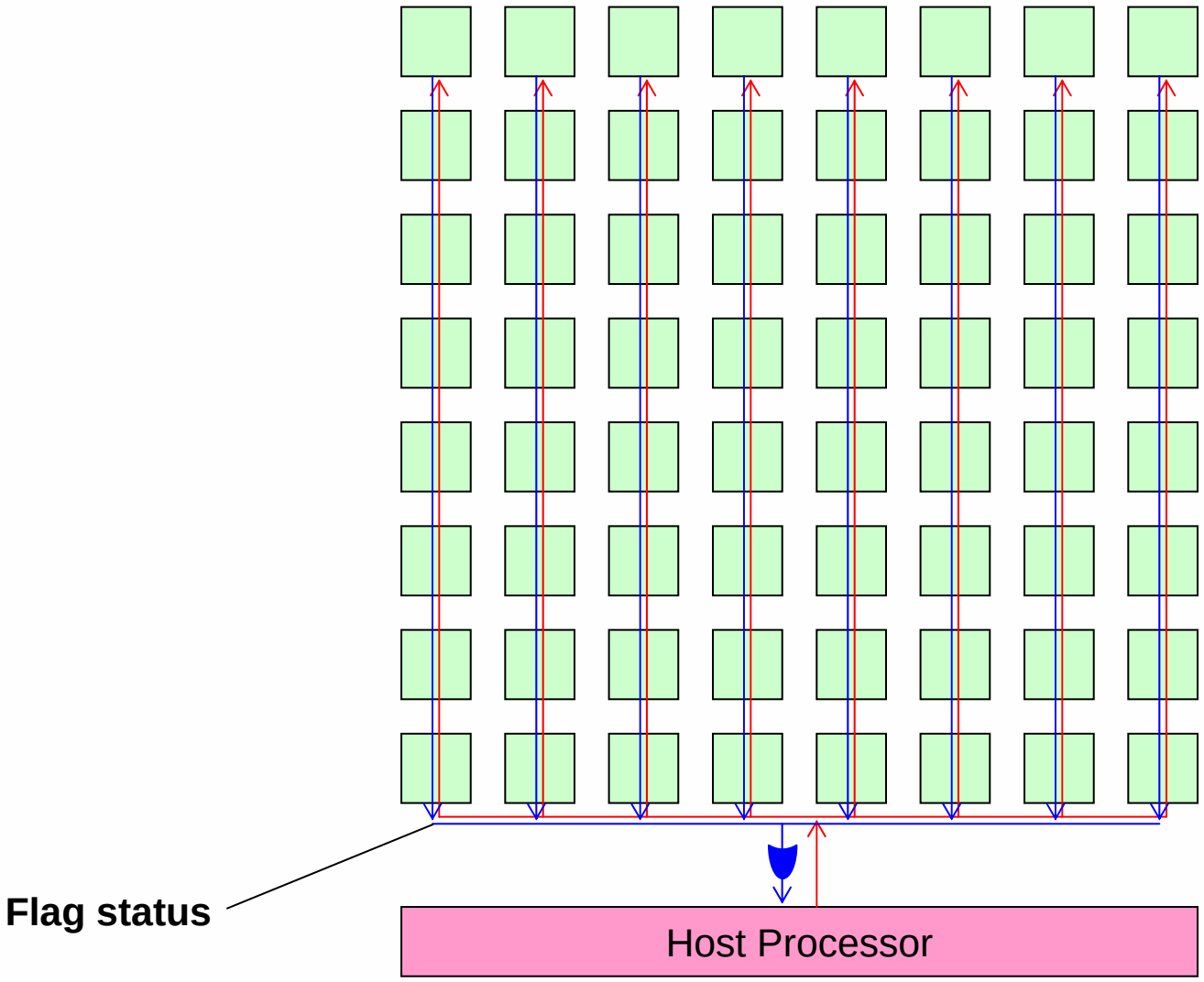
Processing Elements (PEs)



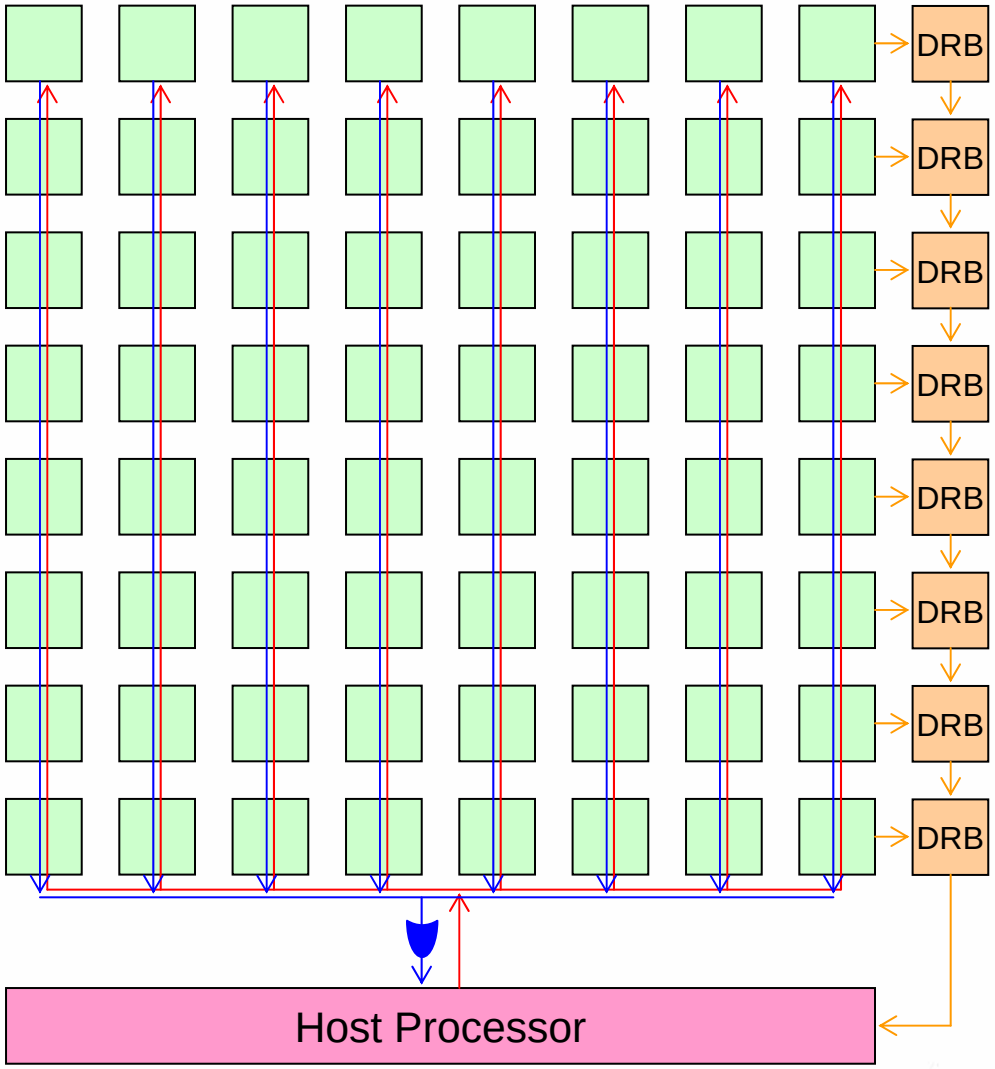
Instruction Bus (IBus)



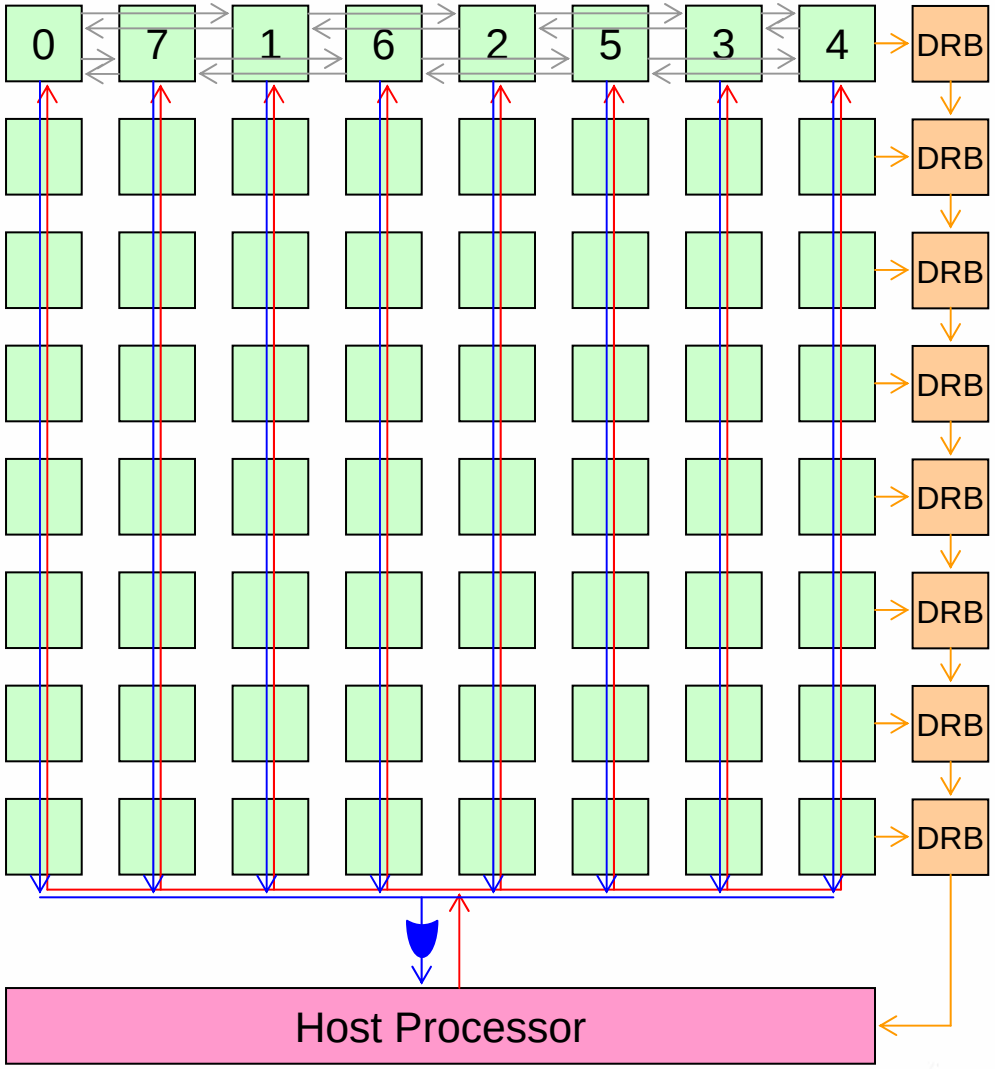
ORTree: Status Monitoring



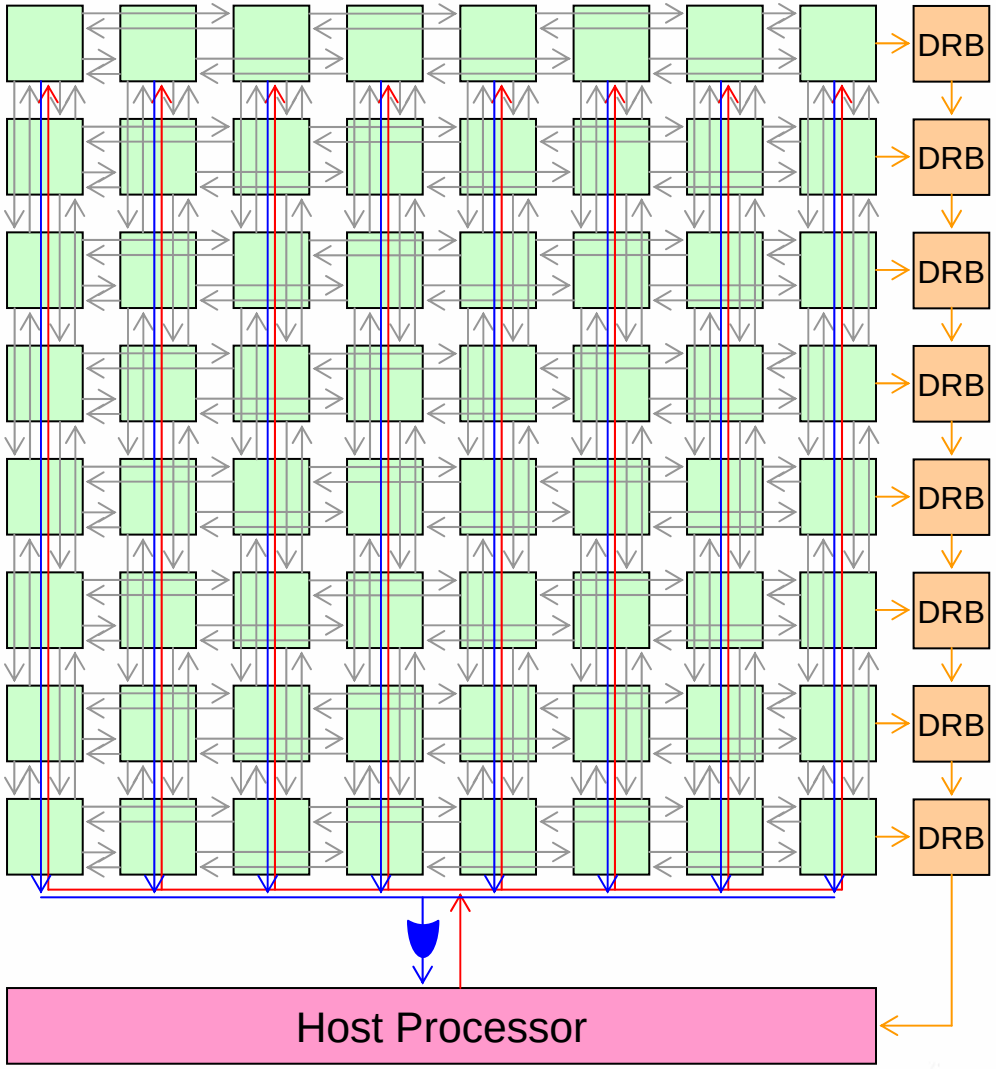
Data Return Buffer (DRB)



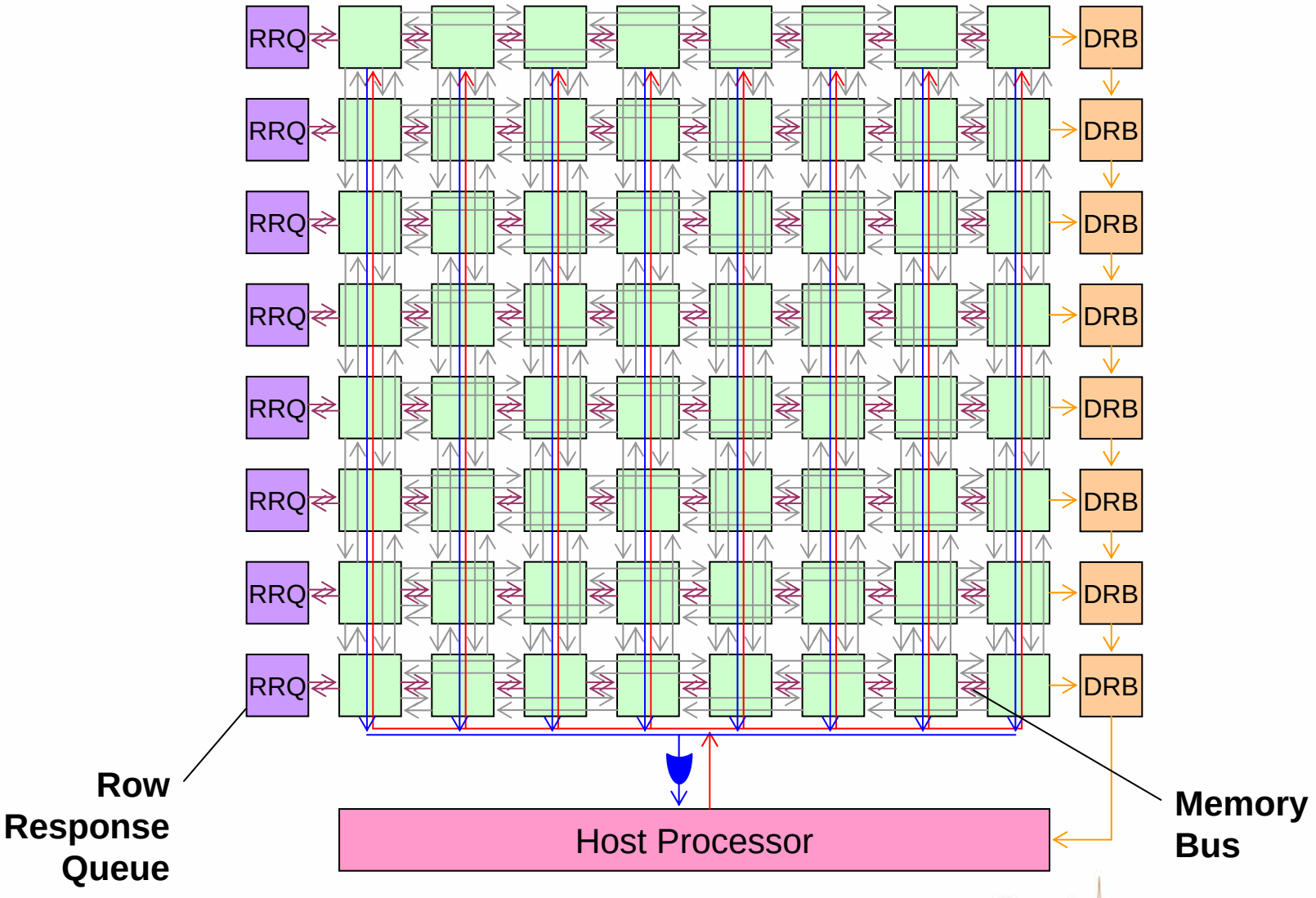
2D Torus P2P Links



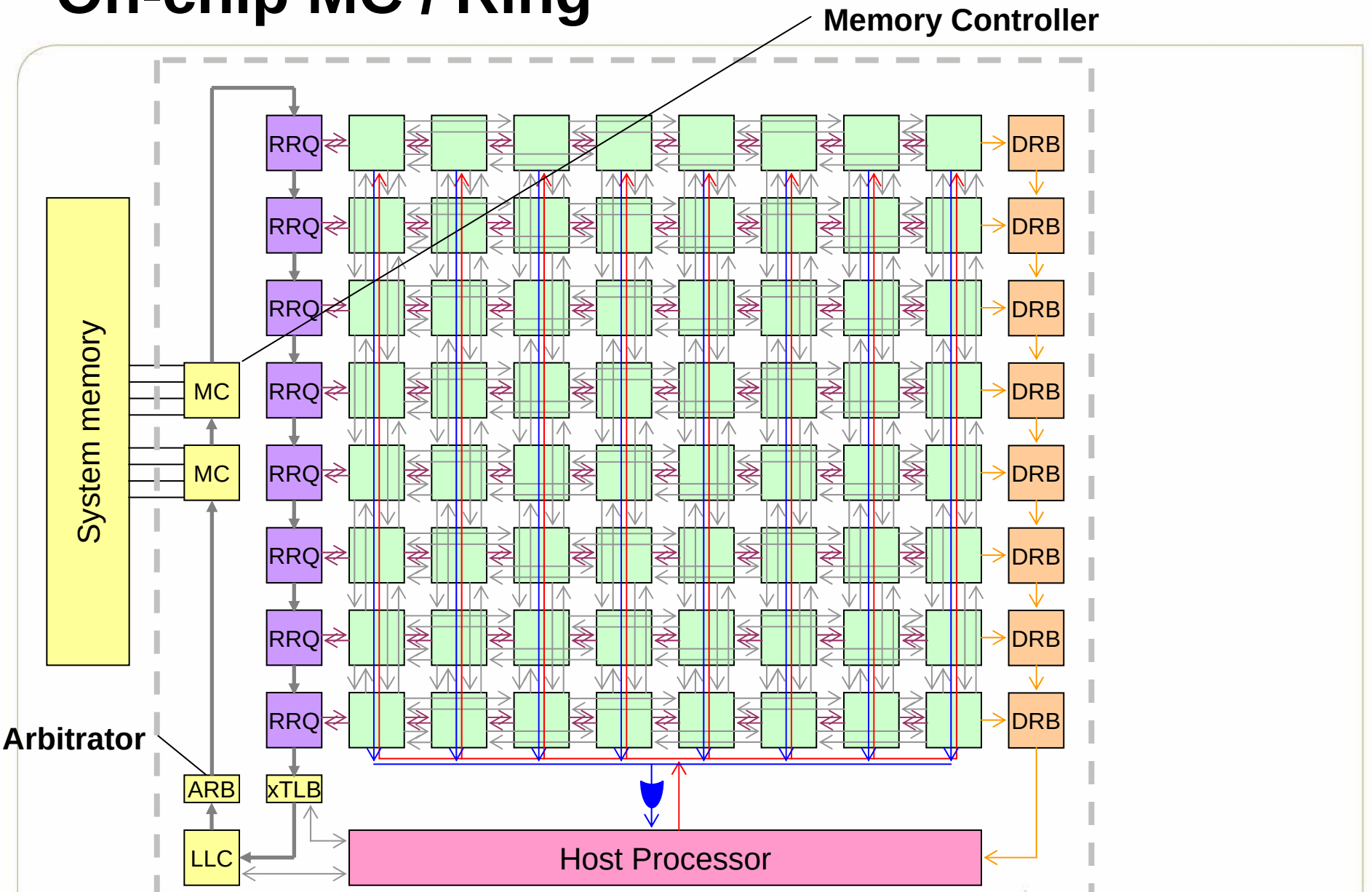
2D Torus P2P Links



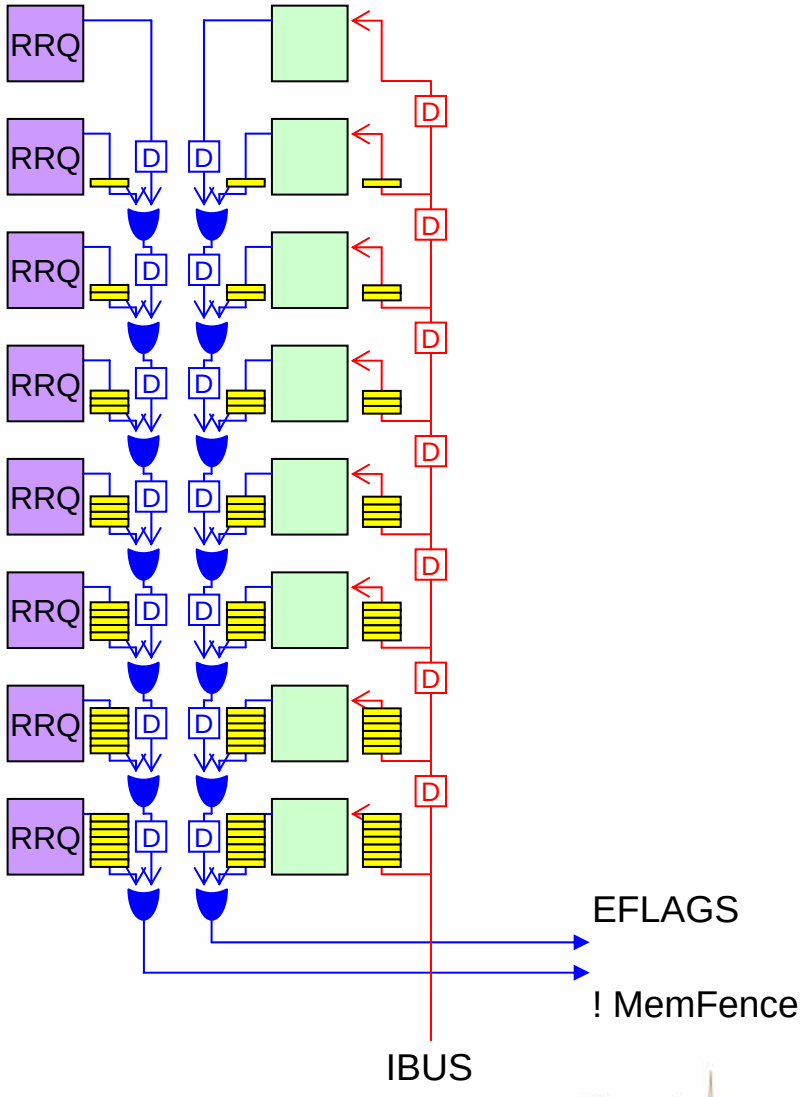
RRQ & MBus: DMA


RRQ
RRQ
RRQ
RRQ
RRQ
RRQ
RRQ
RRQ
DRB
DRB
DRB
DRB
DRB
DRB
DRB
DRB
Row Response Queue
Host Processor
Memory Bus

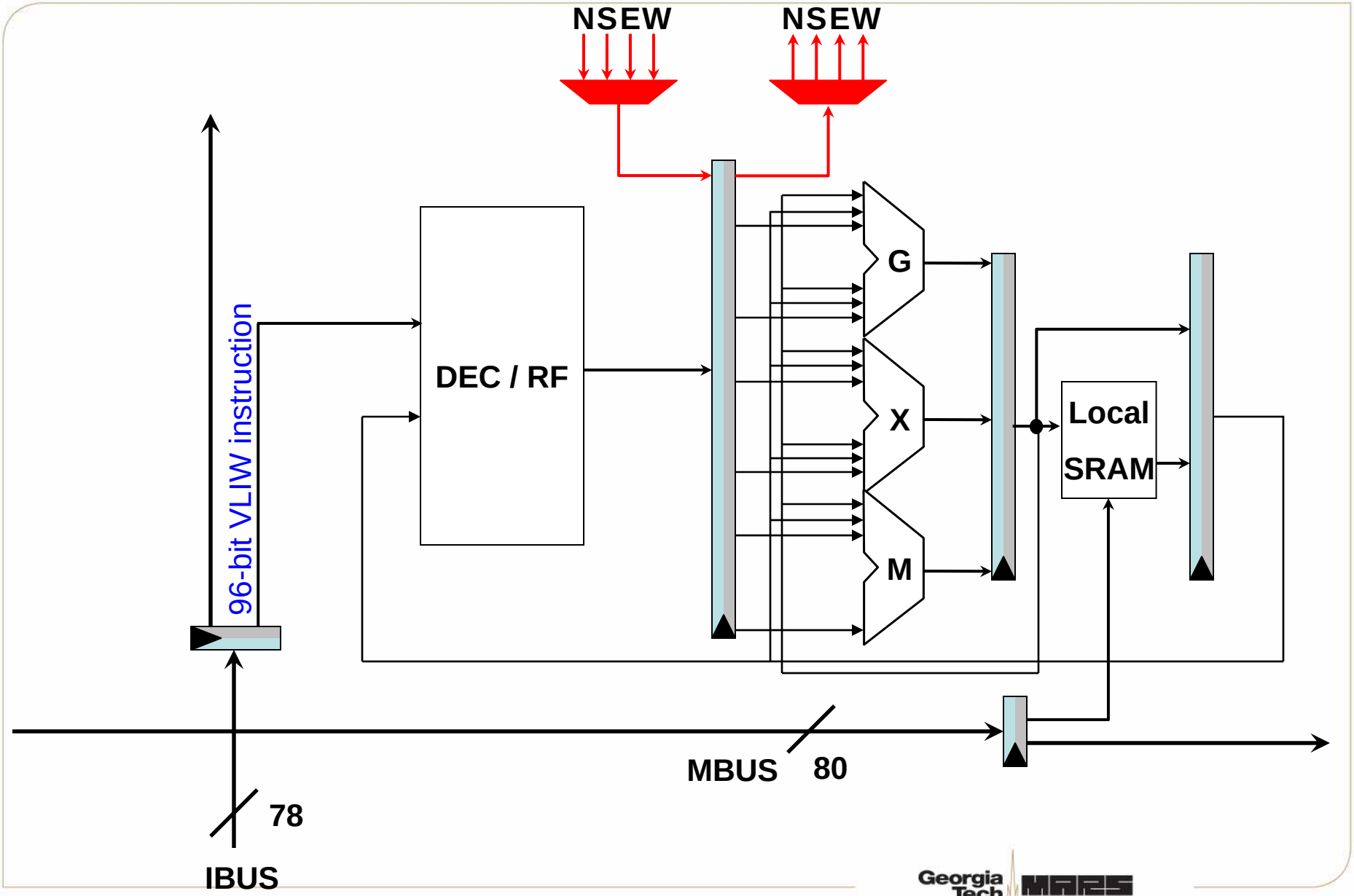
On-chip MC / Ring



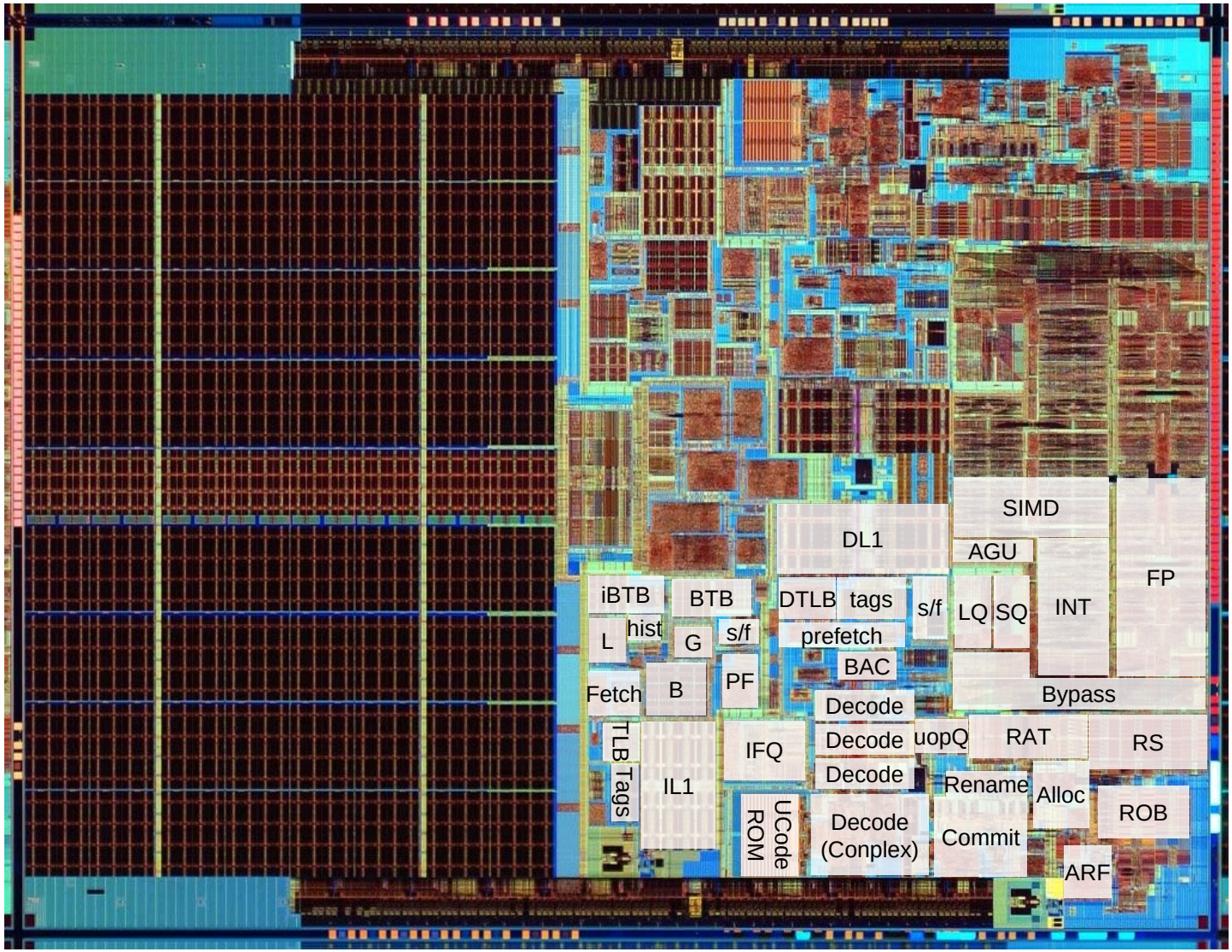
Wire Delay Aware Design



Simplified PE Pipelining

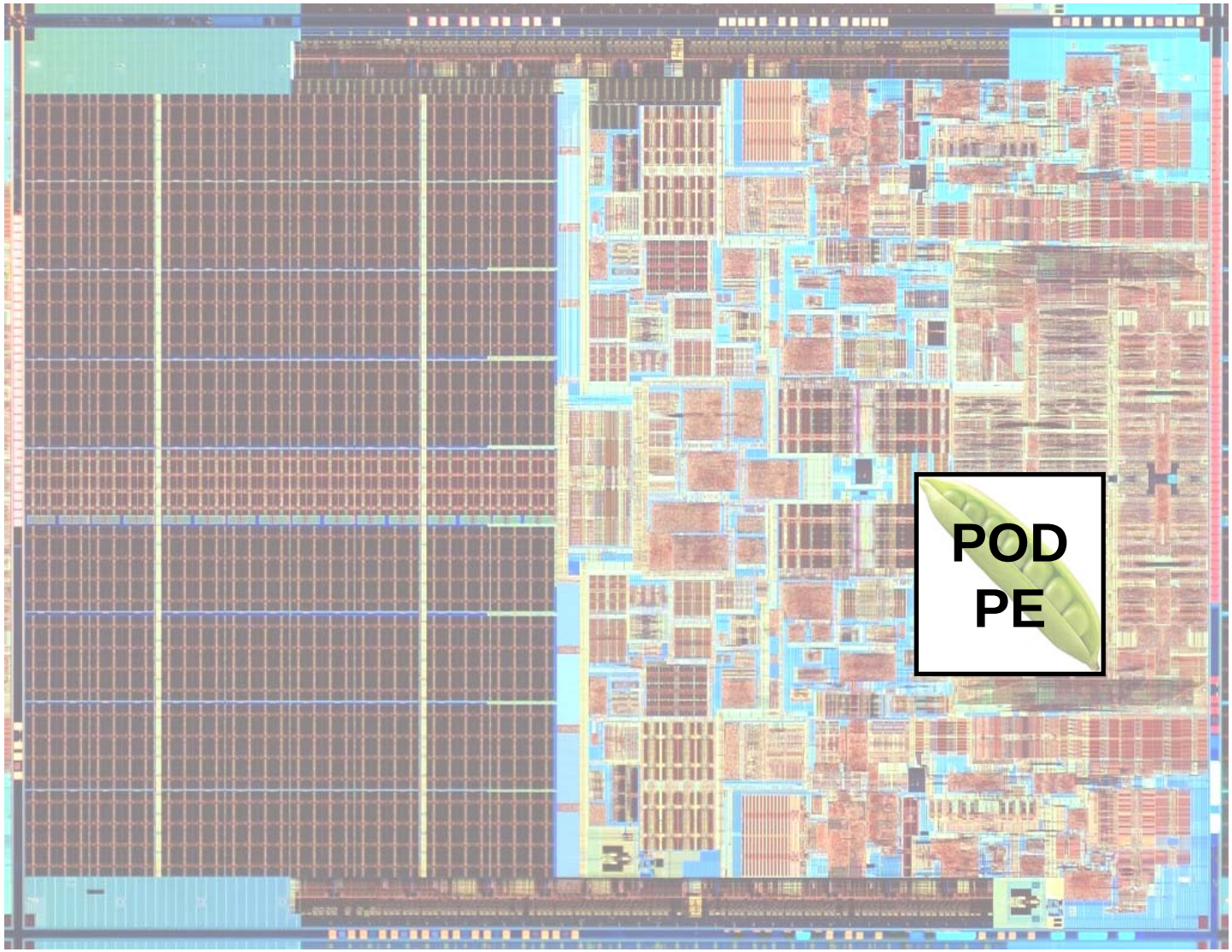


Physical Design Evaluation



Intel Conroe Core 2 Duo processor die photo

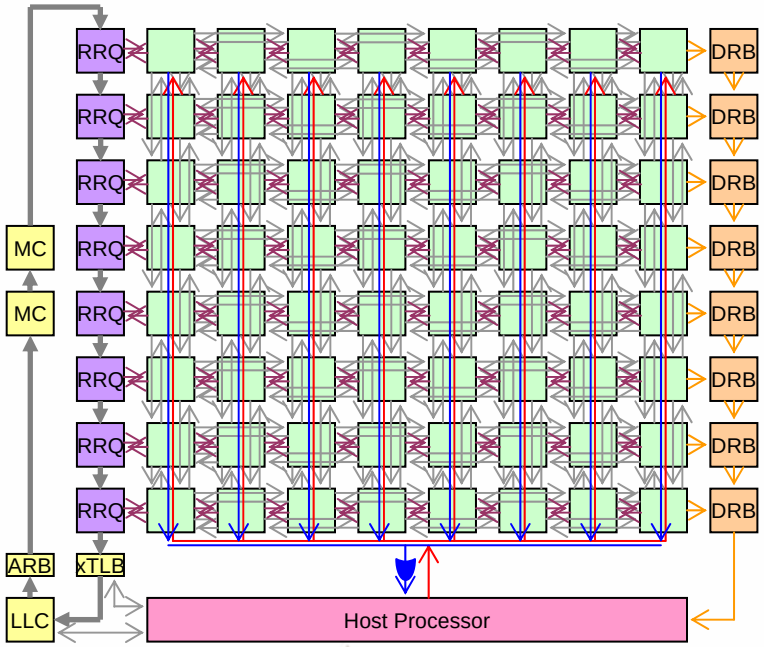
Physical Design Evaluation



Intel Conroe Core 2 Duo processor die photo

Physical Design Evaluation @ 45nm

- **PE**
  - 128KB dual-ported SRAM, Basic GP ALU, simplified SSE ALU
  - ~3.2 mm^2
- **RRQ**
  - ~3.2 mm^2 (conservative estimation)
- **The host processor**
  - Core-based microarchitecture
  - ~25.9 mm^2
- **3MB LLC**
  - ~20 mm^2
- **Total: ~276.5 mm^2**



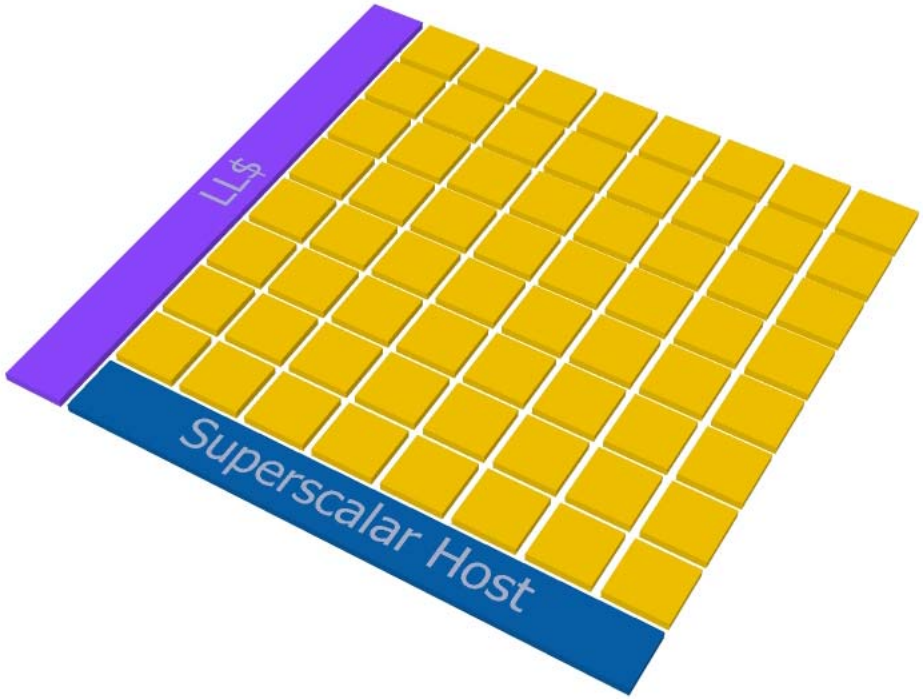
Interconnection Fabric Among PEs

- **Different links for different communication**
  - **IBus / Mbus / 2D torus P2P links / ORTree**
  - **No contention among different communication**

- **Fully synchronized and orchestrated by the host processor (and RRQ for MBus)**
  - **No crossbar**
  - **No contention / arbitration**
  - **No buffer space for the router**
  - **Nothing unpredictable!**

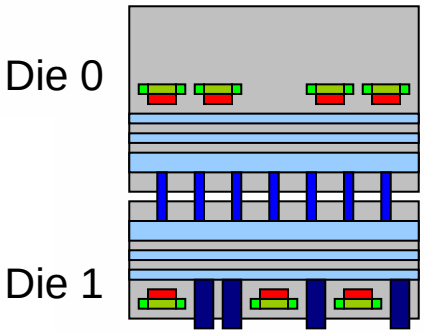
Design Space Exploration

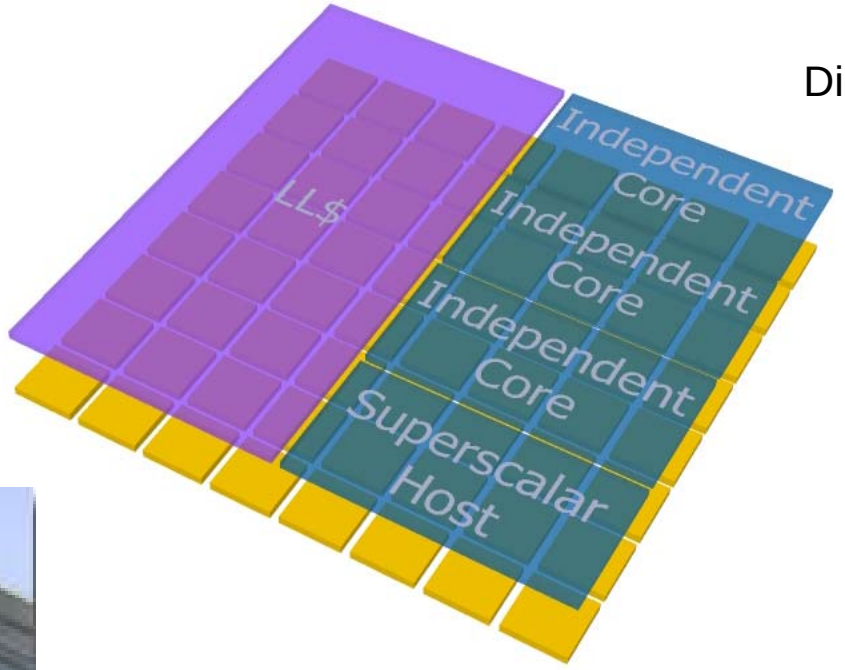
- **Baseline Design**



Design Space Exploration

- **3D POD** with many-core



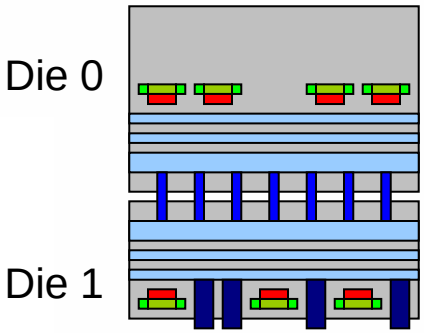


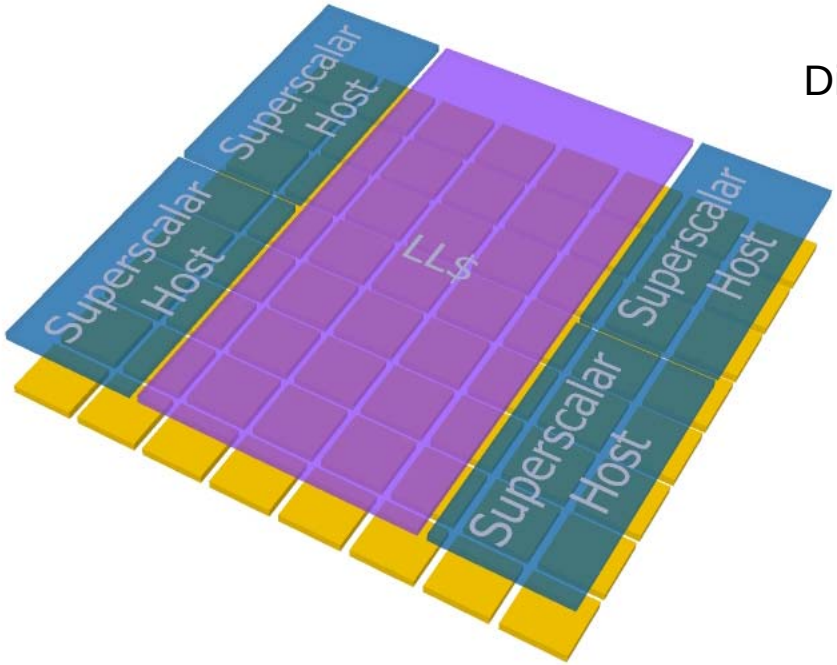
You live and **work** here

You get your **beer** here

Design Space Exploration

- **3D many-POD processor**





Programming Model

Programming Model

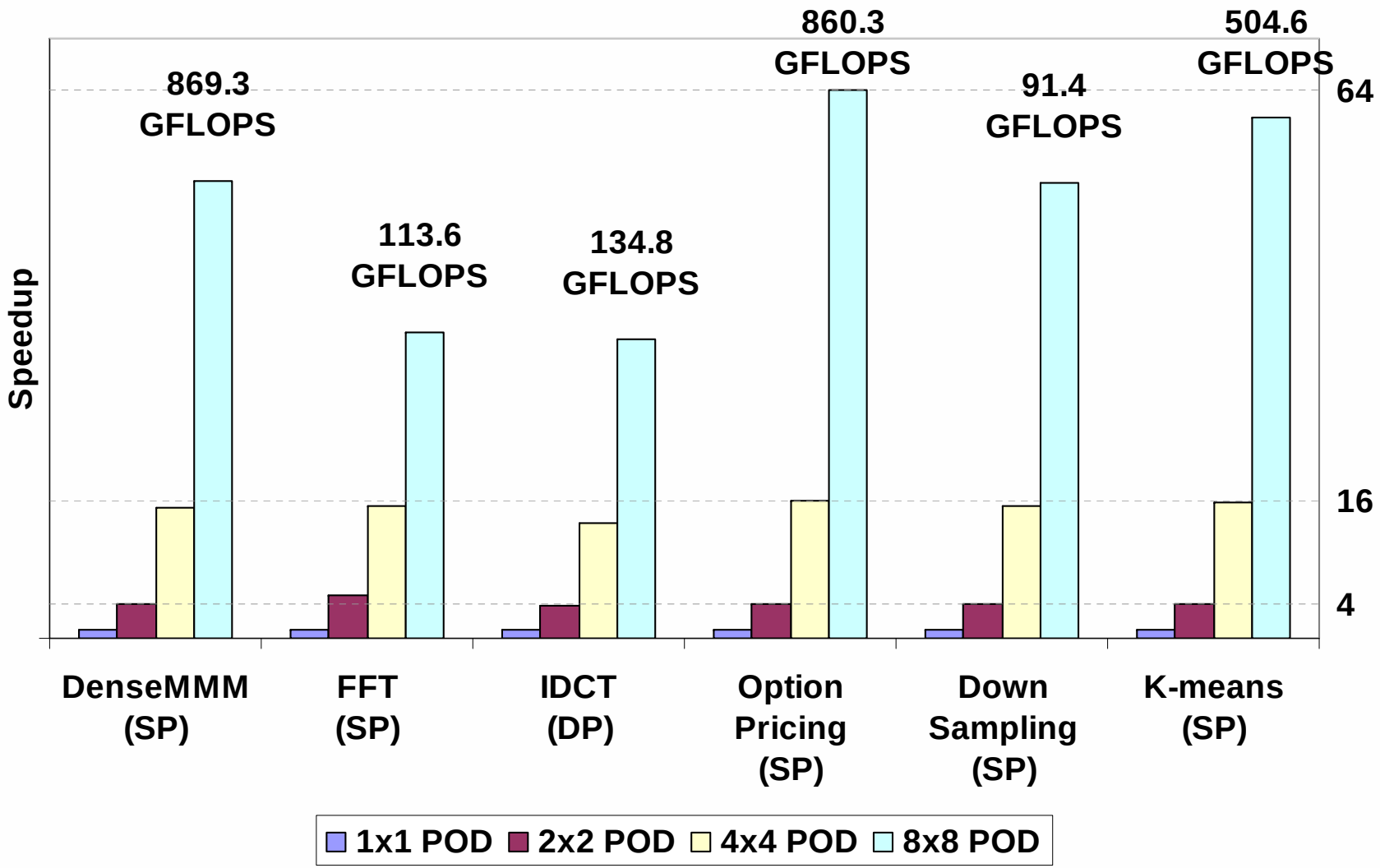
- **Example code: Reduction**

$$S = \sum_{i=0}^{N-1} a_i$$

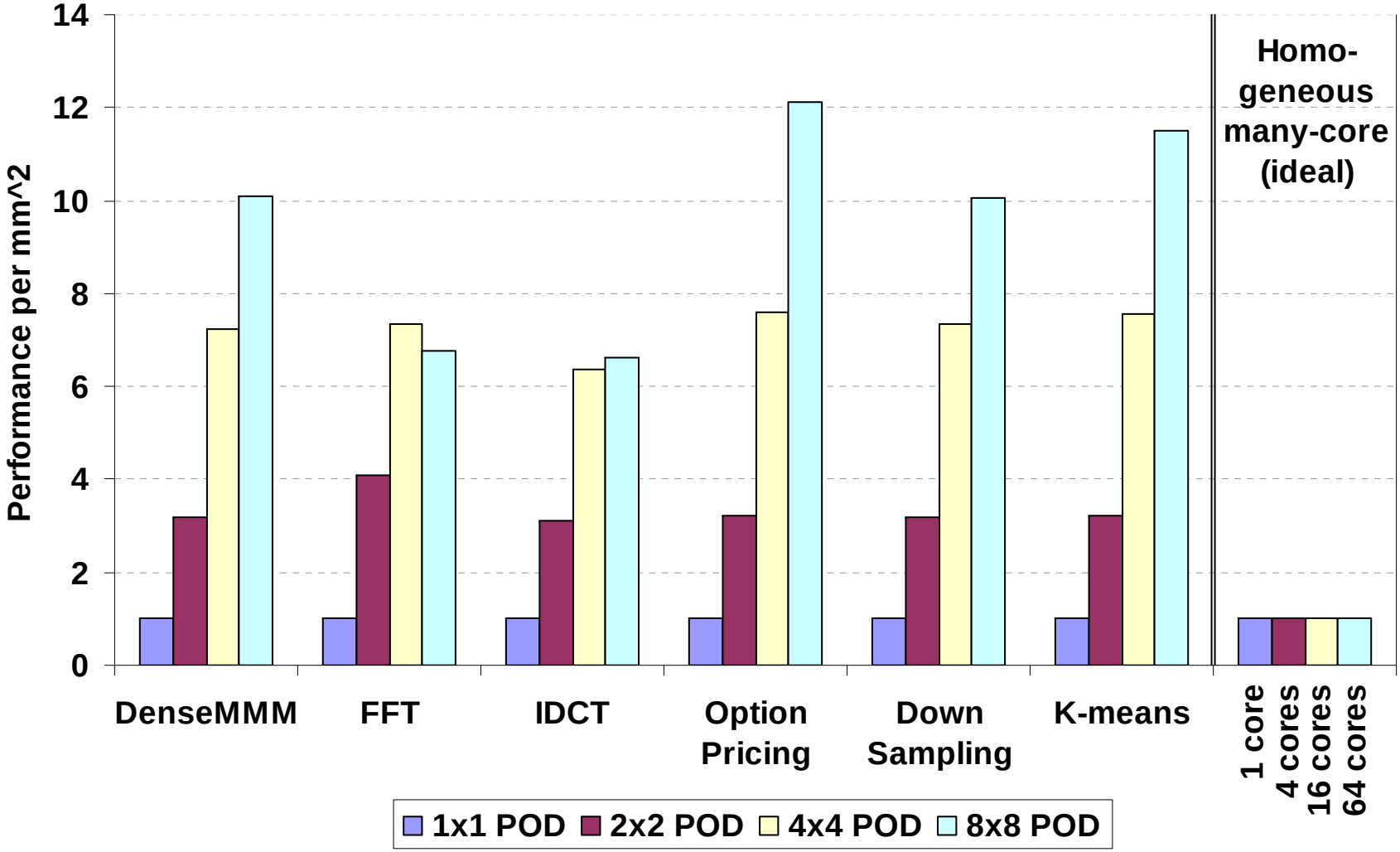
```
char* base_addr = (char*) malloc(npe);
for ( i=0; I < npe; i++ ) {
        *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
        $ASM xfer.east r1 = r1
        $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
        $ASM xfer.north r1 = r1
        $ASM add r2 = r2, r1
}
```

Simulation Result

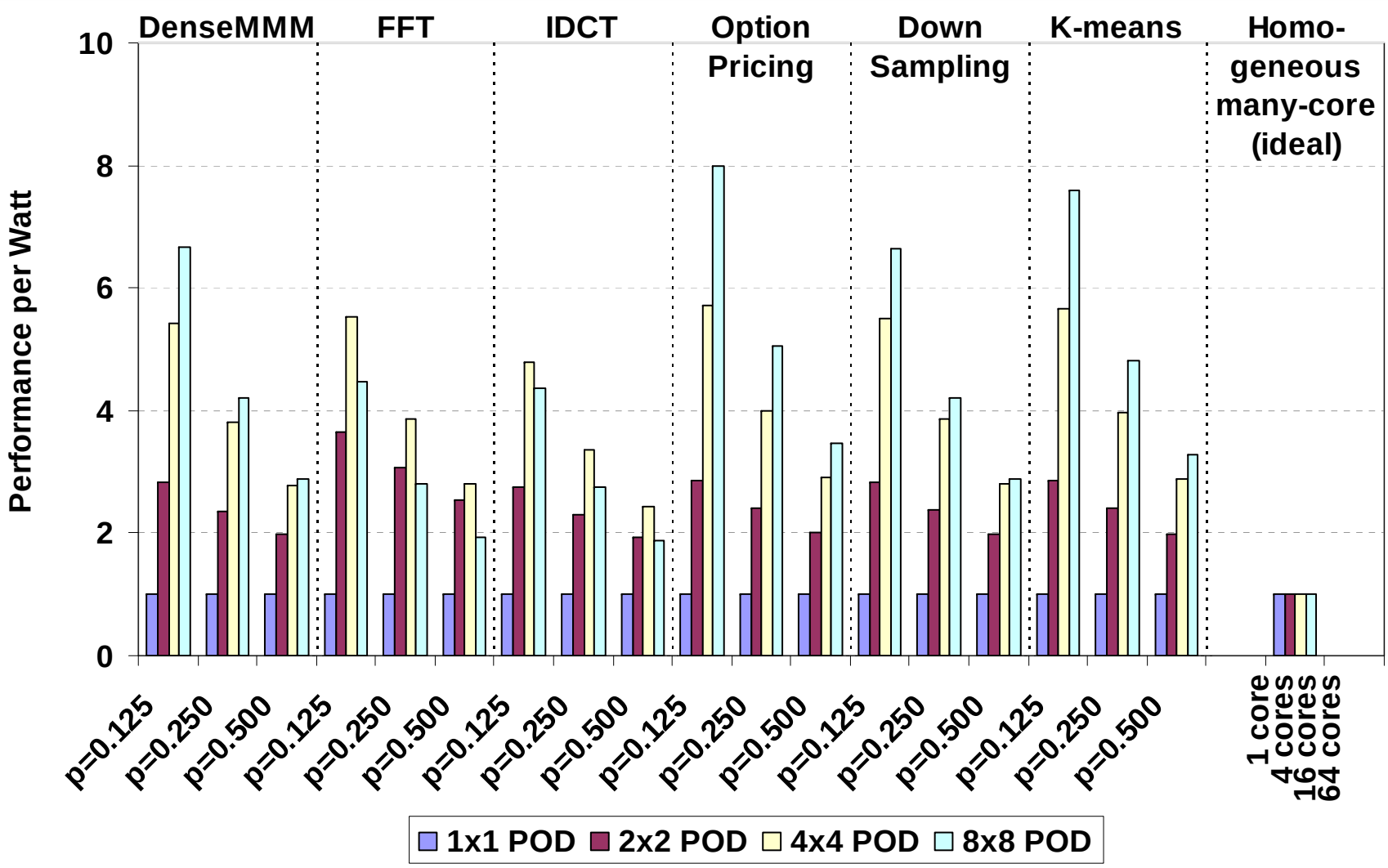
Performance Result



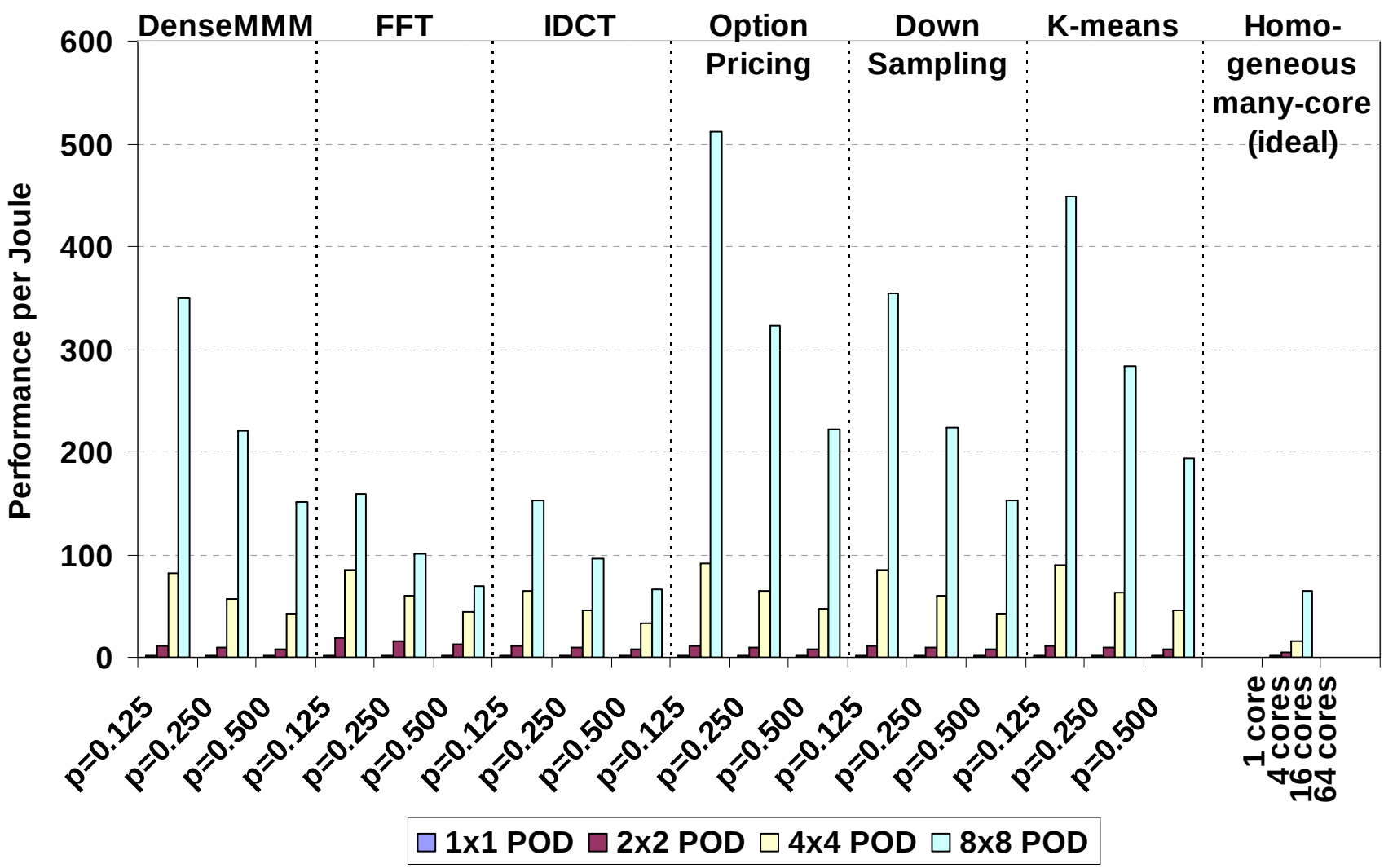
Performance per mm²



Performance per Watt



Performance per Joule

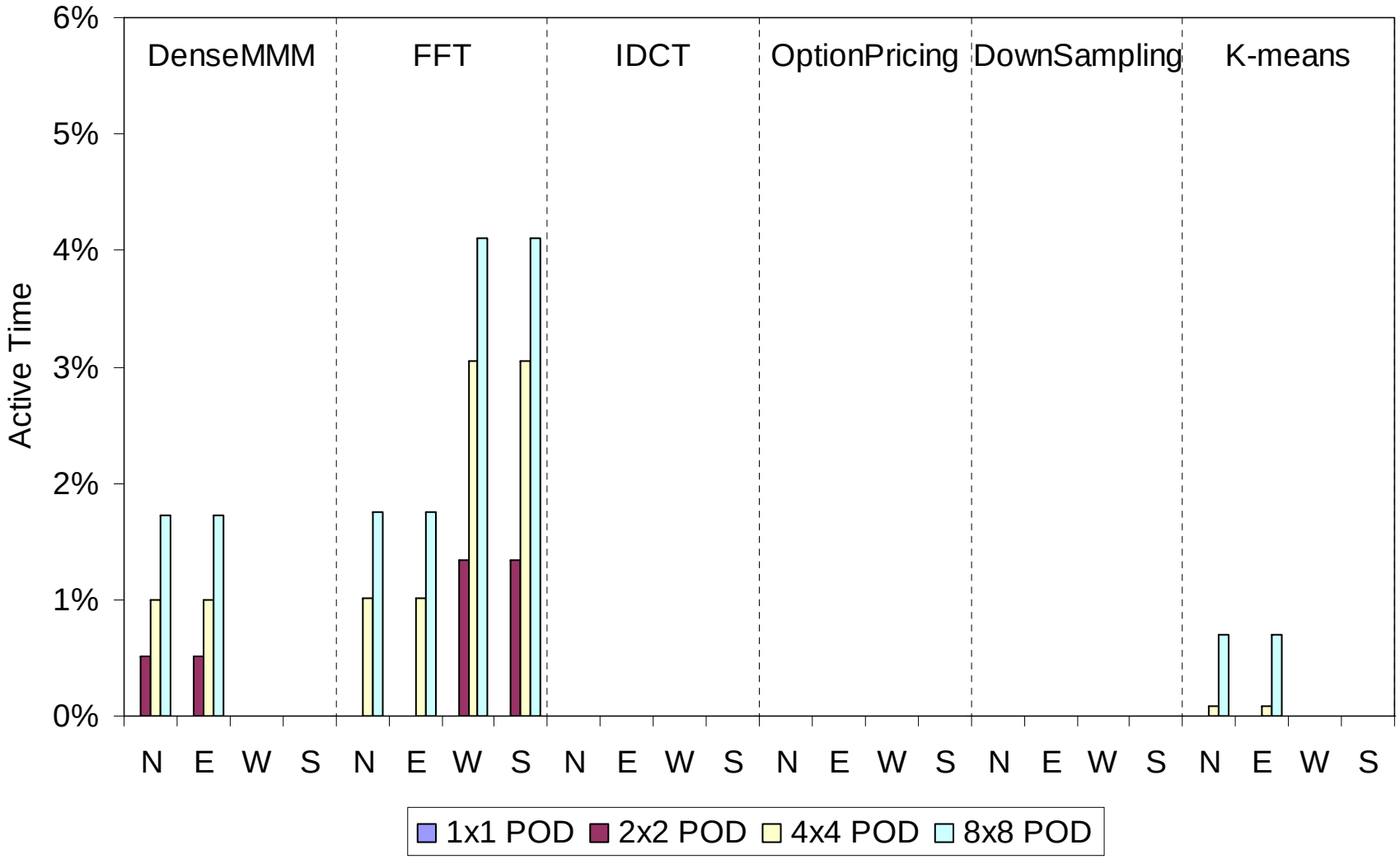


Interconnection Power

	Input buffer	X-bar	arbiter	Link
RAW	31%	30%	~0%	39%
TRIPS	35%	33%	1%	31%

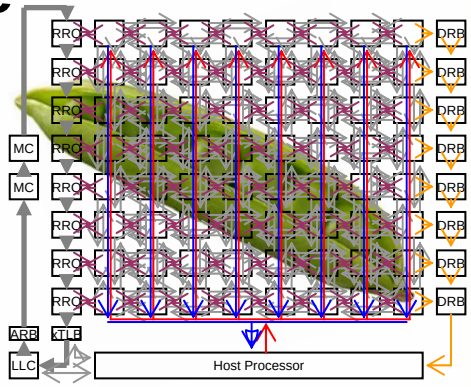
Wang et al., “Power-Driven Design of Router Microarchitectures in On-Chip Networks”, MICRO 2003

2D Torus P2P Link Active Time



Conclusion

- **We propose POD**
  - **Parallel-On-Die many-core architecture**
  - **Broad-purpose acceleration**
  - **High energy and area efficiency**
  - **Backward compatibility**



- **A single-chip POD can achieve up to 1.5 TFLOPS of IEEE SP FP operations.**

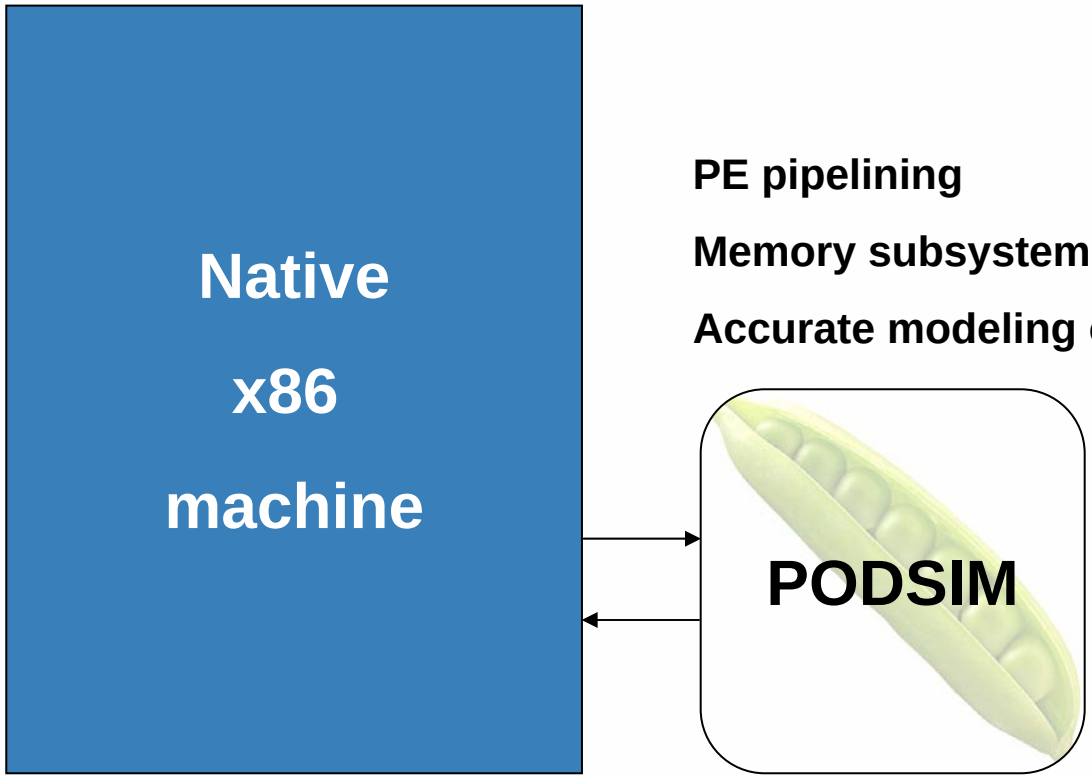
- **Interconnection architecture of POD is extremely power- and area-efficient.**

Let's Use Power to Compute, Not Commute!

Georgia Tech
ECE MARS Labs
http://arch.ece.gatech.edu

Back-up Slides

PODSIM



PE pipelining

Memory subsystem

Accurate modeling of on-chip, off-chip bandwidth

Limitation:

Host processor overhead not modeled (I$ miss, branch misprediction)

LLC takeover of the MC ring by the host

Why multi-processing again?

- No more faster processor
  - Power wall
  - Increasing wi
  - Diminishing r lined architecture
- No more illusi l processor
  - Design comp
  - Diminishing return of wide-issue superscalar processors

Out-of-order Host Issues

- **Speculative execution**
  - **No recovery mechanism in each PE**
  - **Broadcast POD instructions in a non-speculative manner**
    - **As long as host-side code does not depend on the results from POD, this approach does not affect the performance.**
- **Instruction Reordering**
  - **POD instructions are strongly ordered.**
    - **IBits queue**
      - **Similar to the store queue**

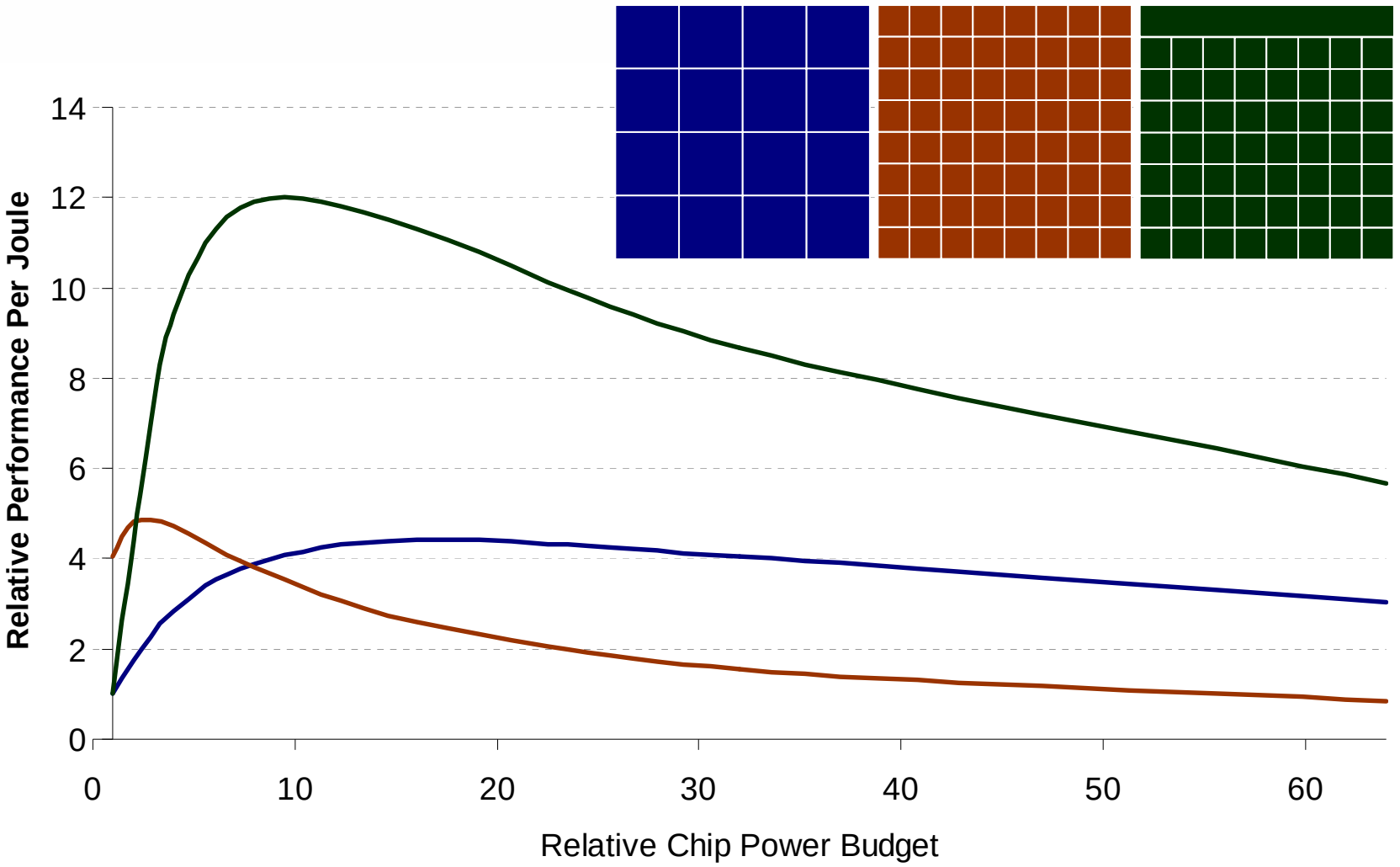
x86 Modification

- 5 new instructions:
  - sendibits, getort, drainort, movl, getdrb,
- 3 modified instructions:
  - lfence, sfence, mfence
- 4 possible new co-processor registers or else mem-mapped addrs:
  - ibits register, ort status, drb value, xTLB support
- IBits queue
- All other modifications external to the x86 core
  - Arbiter for LLC priority can be external to LLC
- Deferred Design Issue: LLC-POD Coherence
  - v1 Study: uncachable
  - v2 Prototype: LLC Snoops on A/D rings

POD ISA

- "Instructions" are 3-wide VLIW bundles,
  - One of each G,X,M (32 bits each)
- Integer Pipe (G):
  - and,or,not,xor,shl,shr,add,sub,mul,cmp,cmov,xfer,xferdrb : !div
- SSE Pipe (X):
  - pfp{fma,add,sub,mul,div,max,min},pi{add,sub,mul,and,or,not,cmp},shuffle,cvt,xfer : !idiv
- Memory Pipe (M):
  - ld,st,blkrd/wr,mask{set,pop,push},mov,xfer
- Hybrid (G+X+M):
  - movl

Power Scalability!



Some Known Facts on Interconnection

- **MIT RAW**
  - **~ 40% of die area**

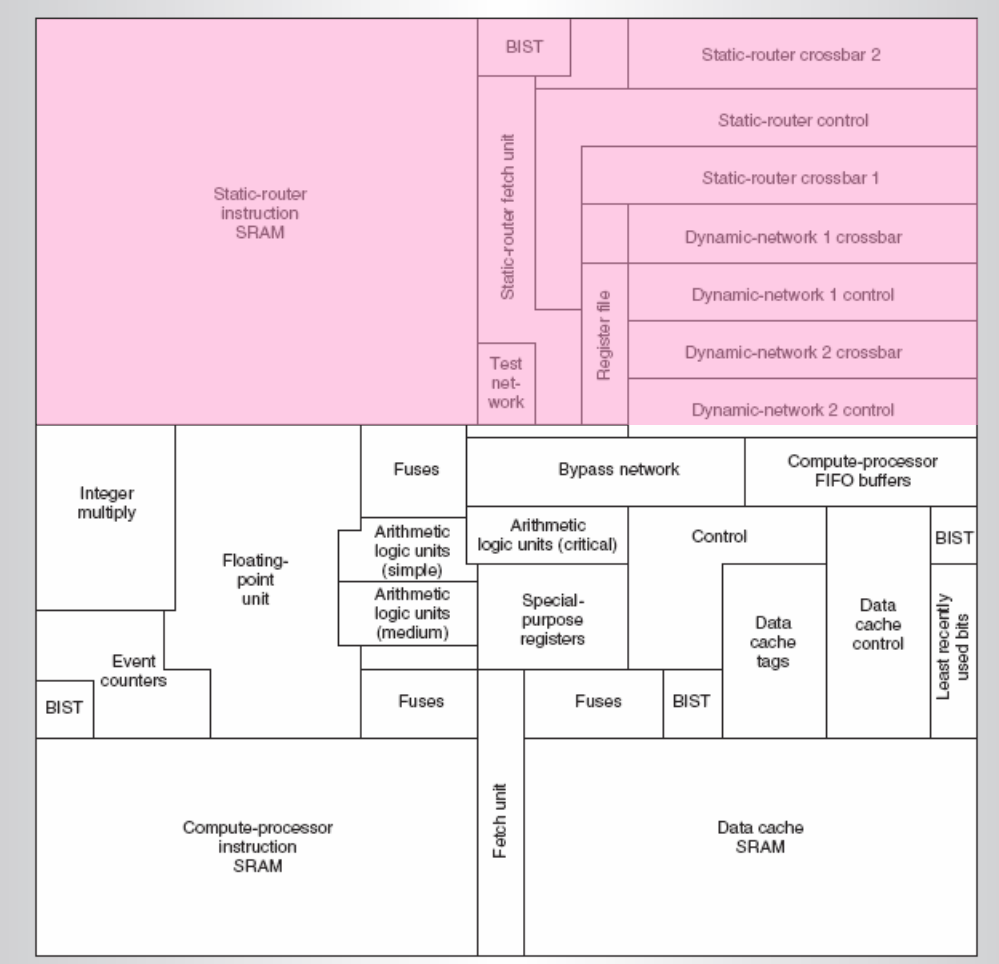


Figure 8. Raw tile floor plan.

Taylor et al., "The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs", IEEE MICRO Mar/Apr 2002

Some Known Facts on Interconnection

- **One main energy hog!**
  - **Alpha 21364:**
    - **20% (25W / 125W)**
  - **MIT RAW:**
    - **40% of individual tile power**
  - **UT TRIPS:**
    - **25%**

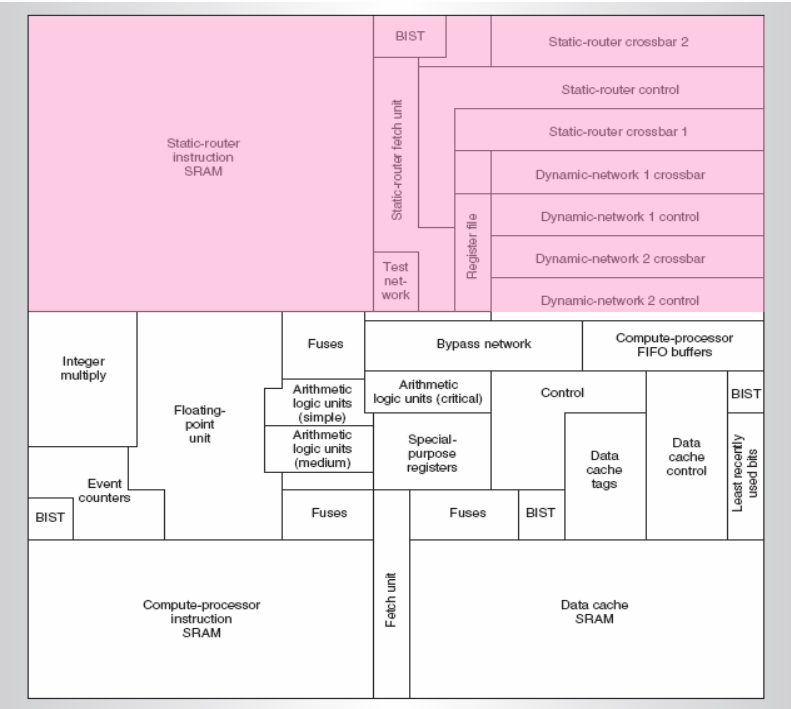


Figure 8. Raw tile floor plan.

Taylor et al., "The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs", IEEE MICRO Mar/Apr 2002
Wang et al., "Orion: A Power-Performance Simulator for Interconnection Networks", MICRO 2002
Wang et al., "Power-Driven Design of Router Microarchitectures in On-Chip Networks", MICRO 2003
Peh, "Chip-Scale Power Scaling", http://www.princeton.edu/~peh/talks/stanford_nws.pdf

Some Known Facts on Interconnection

- **Mesh-based MIMD many-core?**
  - **Unsustainable energy in its router logic**
  - **Unpredictable communication pattern**
    - **Difficult to apply common low-power techniques**

Bokar, “Networks for Multi-core Chip—A Controversial View”, 2006 Workshop on On- and Off-Chip Interconnection Networks for Multicore Systems

Programming Model

Programming Model

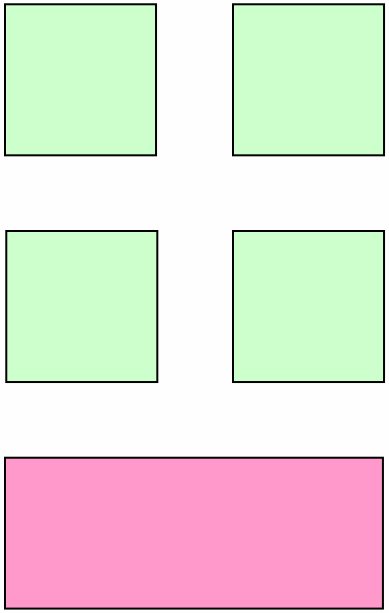
- **Example code: Reduction**

$$S = \sum_{i=0}^{N-1} a_i$$

```
char* base_addr = (char*) malloc(npe);
for ( i=0; I < npe; i++ ) {
        *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
        $ASM xfer.east r1 = r1
        $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
        $ASM xfer.north r1 = r1
        $ASM add r2 = r2, r1
}
```

Programming Model (2x2 POD)

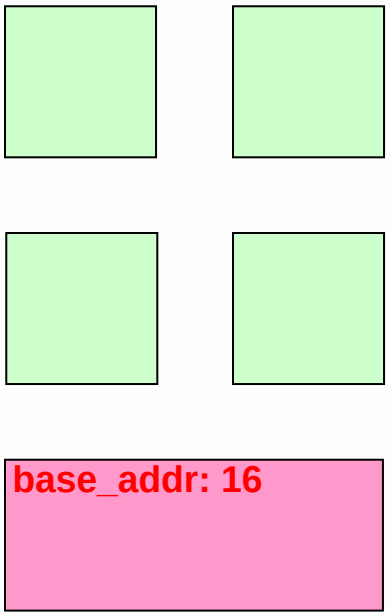
- **Malloc memory at the host memory space**



```
char* base_addr = (char*) malloc(npe);
for ( i=0; I < npe; i++ ) {
        *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
        $ASM xfer.east r1 = r1
        $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
        $ASM xfer.north r1 = r1
        $ASM add r2 = r2, r1
}
```

Programming Model (2x2 POD)

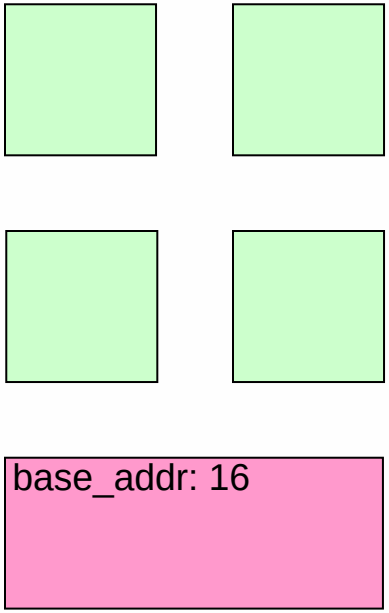
- **Initialize a_i**




```
char* base_addr = (char*) malloc(npe);
for ( i=0; I < npe; i++ ) {
        *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
        $ASM xfer.east r1 = r1
        $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
        $ASM xfer.north r1 = r1
        $ASM add r2 = r2, r1
}
```

Programming Model (2x2 POD)

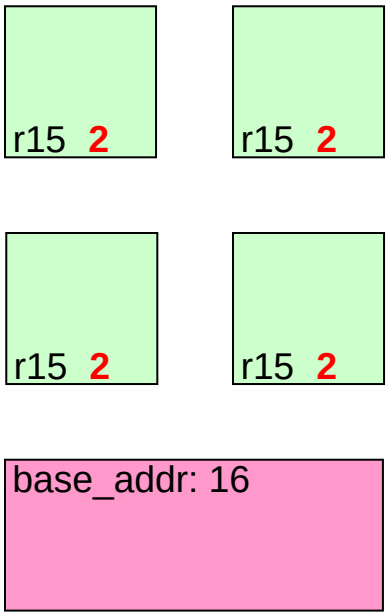
- **Load the address of my PE ID**




```
char* base_addr = (char*) malloc(npe);
for ( i=0; I < npe; i++ ) {
        *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
        $ASM xfer.east r1 = r1
        $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
        $ASM xfer.north r1 = r1
        $ASM add r2 = r2, r1
}
```

Programming Model (2x2 POD)

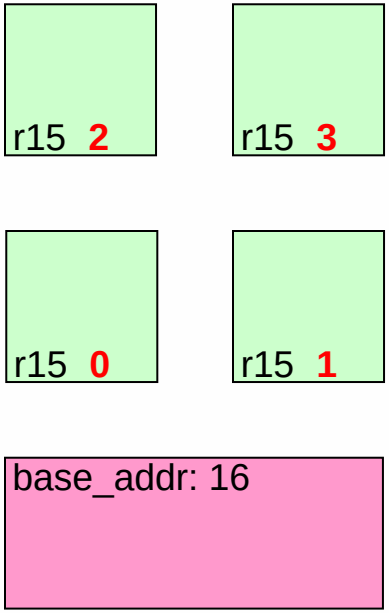
- **Load my PE ID**




```
char* base_addr = (char*) malloc(npe);
for ( i=0; I < npe; i++ ) {
        *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
        $ASM xfer.east r1 = r1
        $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
        $ASM xfer.north r1 = r1
        $ASM add r2 = r2, r1
}
```

Programming Model (2x2 POD)

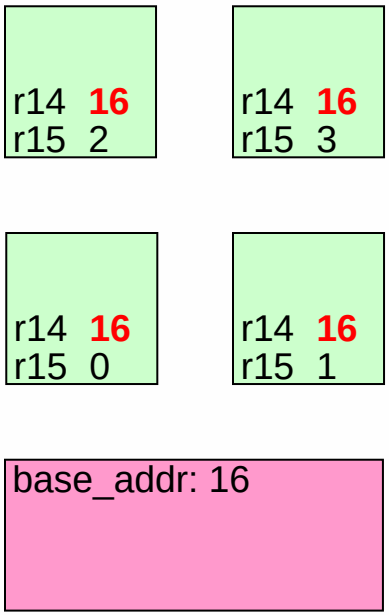
- **Broadcast base_addr**




```
char* base_addr = (char*) malloc(npe);
for ( i=0; I < npe; i++ ) {
        *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
        $ASM xfer.east r1 = r1
        $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
        $ASM xfer.north r1 = r1
        $ASM add r2 = r2, r1
}
```

Programming Model (2x2 POD)

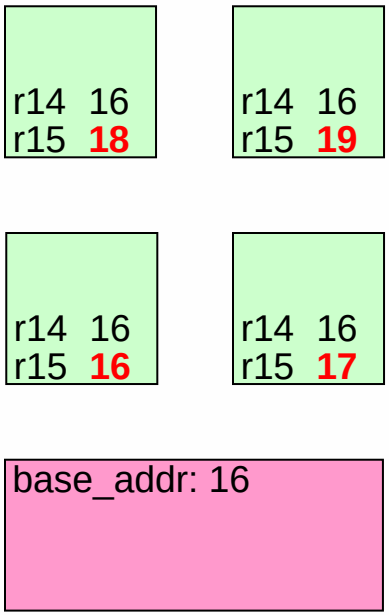
- **Calculate the pointer to my a_i**




```
char* base_addr = (char*) malloc(npe);
for ( i=0; I < npe; i++ ) {
        *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
        $ASM xfer.east r1 = r1
        $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
        $ASM xfer.north r1 = r1
        $ASM add r2 = r2, r1
}
```

Programming Model (2x2 POD)

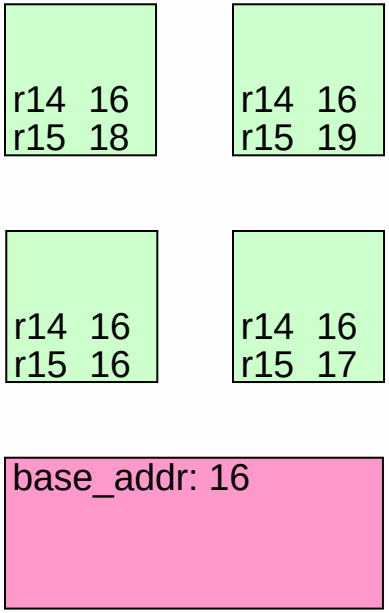
- **Load my a_i from the host memory space**




```
char* base_addr = (char*) malloc(npe);
for ( i=0; I < npe; i++ ) {
        *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

➡ $ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
        $ASM xfer.east r1 = r1
        $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
        $ASM xfer.north r1 = r1
        $ASM add r2 = r2, r1
}
```

Programming Model (2x2 POD)

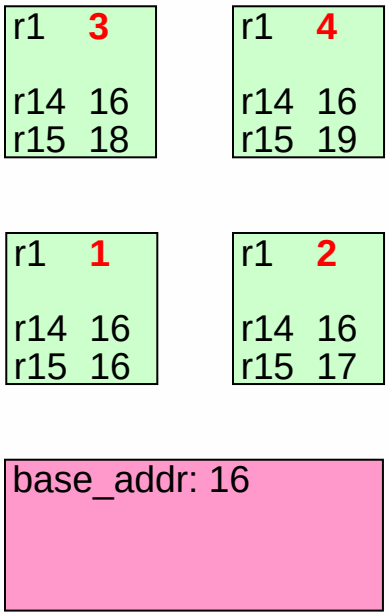
- **Wait until it is loaded**




```
char* base_addr = (char*) malloc(npe);
for ( i=0; I < npe; i++ ) {
        *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
        $ASM xfer.east r1 = r1
        $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
        $ASM xfer.north r1 = r1
        $ASM add r2 = r2, r1
}
```

Programming Model (2x2 POD)

- **Update *S***




```
char* base_addr = (char*) malloc(npe);
for ( i=0; I < npe; i++ ) {
        *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
        $ASM xfer.east r1 = r1
        $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
        $ASM xfer.north r1 = r1
        $ASM add r2 = r2, r1
}
```

Programming Model (2x2 POD)

- 1st iteration

r1	3
r2	**3**
r14	16
r15	18

r1	4
r2	**4**
r14	16
r15	19

r1	1
r2	**1**
r14	16
r15	16

r1	2
r2	**2**
r14	16
r15	17

base_addr: 16

16: 1
17: 2
18: 3
19: 4

```
char* base_addr = (char*) malloc(npe);
for ( i=0; I < npe; i++ ) {
        *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
        $ASM xfer.east r1 = r1
        $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
        $ASM xfer.north r1 = r1
        $ASM add r2 = r2, r1
}
```

Programming Model (2x2 POD)

- **Transfer my a_i to my east-side neighbor**

r1	3
r2	3
r14	16
r15	18

r1	4
r2	4
r14	16
r15	19

r1	1
r2	1
r14	16
r15	16

r1	2
r2	2
r14	16
r15	17

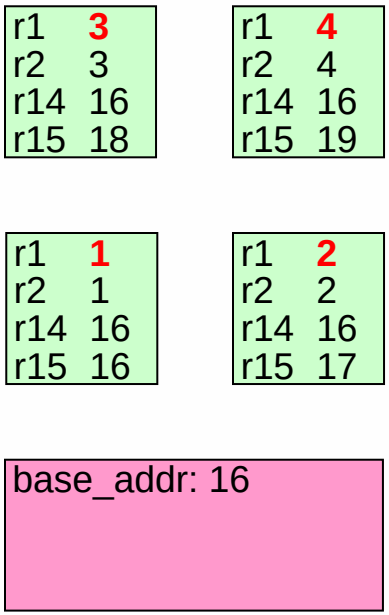
base_addr: 16

16: 1
17: 2
18: 3
19: 4

```
char* base_addr = (char*) malloc(npe);
for ( i=0; I < npe; i++ ) {
        *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
        $ASM xfer.east r1 = r1
        $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
        $ASM xfer.north r1 = r1
        $ASM add r2 = r2, r1
}
```

Programming Model (2x2 POD)

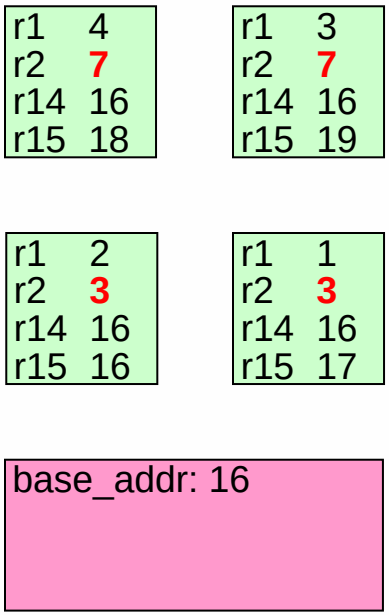
- **Update *S***




```
char* base_addr = (char*) malloc(npe);
for ( i=0; I < npe; i++ ) {
        *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
        $ASM xfer.east r1 = r1
        $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
        $ASM xfer.north r1 = r1
        $ASM add r2 = r2, r1
}
```

Programming Model (2x2 POD)

- **Copy rowsum to r1**




```
char* base_addr = (char*) malloc(npe);
for ( i=0; I < npe; i++ ) {
        *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
        $ASM xfer.east r1 = r1
        $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
        $ASM xfer.north r1 = r1
        $ASM add r2 = r2, r1
}
```

Programming Model (2x2 POD)

- 1st iteration

r1	7
r2	7
r14	16
r15	18

r1	7
r2	7
r14	16
r15	19

r1	3
r2	3
r14	16
r15	16

r1	3
r2	3
r14	16
r15	17

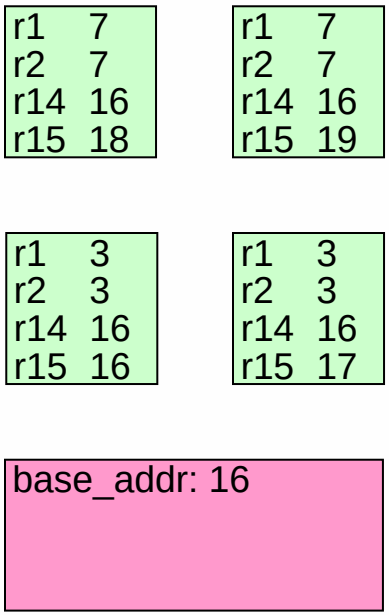
base_addr: 16

16: 1
17: 2
18: 3
19: 4

```
char* base_addr = (char*) malloc(npe);
for ( i=0; I < npe; i++ ) {
        *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
        $ASM xfer.east r1 = r1
        $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
        $ASM xfer.north r1 = r1
        $ASM add r2 = r2, r1
}
```

Programming Model (2x2 POD)

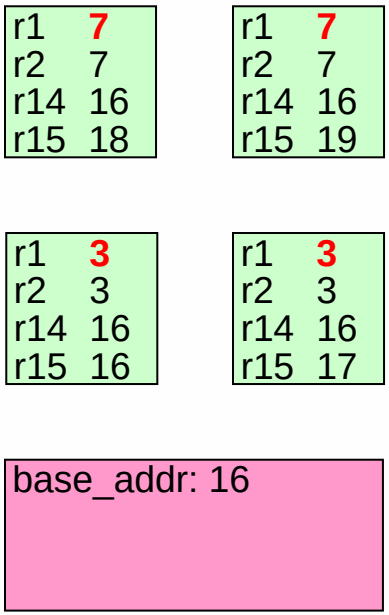
- **Transfer my rowsum to my north-side neighbor**




```
char* base_addr = (char*) malloc(npe);
for ( i=0; I < npe; i++ ) {
        *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
        $ASM xfer.east r1 = r1
        $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
        $ASM xfer.north r1 = r1
        $ASM add r2 = r2, r1
}
```

Programming Model (2x2 POD)

- **Update *S***




```
char* base_addr = (char*) malloc(npe);
for ( i=0; I < npe; i++ ) {
        *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
        $ASM xfer.east r1 = r1
        $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
        $ASM xfer.north r1 = r1
        $ASM add r2 = r2, r1
}
```

Programming Model (2x2 POD)

- **Done!**

r1	3
r2	**10**
r14	16
r15	18

r1	3
r2	**10**
r14	16
r15	19

r1	7
r2	**10**
r14	16
r15	16

r1	7
r2	**10**
r14	16
r15	17

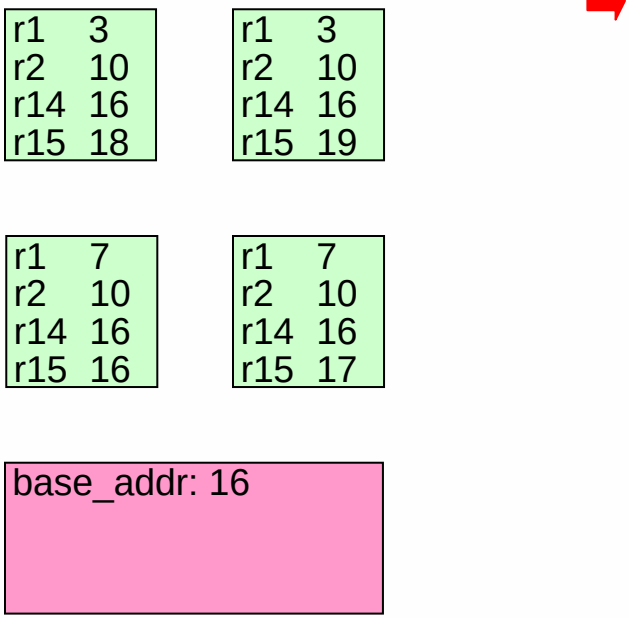
base_addr: 16

16: 1
17: 2
18: 3
19: 4

```
char* base_addr = (char*) malloc(npe);
for ( i=0; I < npe; i++ ) {
        *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
        $ASM xfer.east r1 = r1
        $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
        $ASM xfer.north r1 = r1
        $ASM add r2 = r2, r1
}
```

Programming Model (2x2 POD)

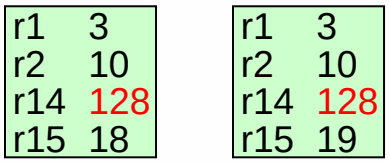
- **Broadcast the pointer to ‘result’**

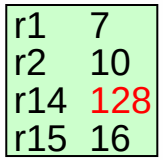


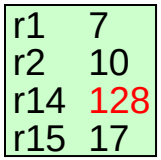

```
int result;

POD_movl( r14, &result );

$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]

$ASM sub r15 = r15, 0
$ASM pushmask.e
        $ASM st sys[ r14 + 0 ] = r15
$ASM popmask

POD_mfence();
```

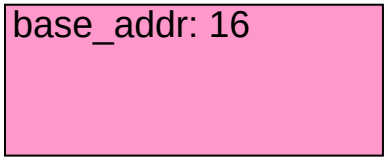
Programming Model (2x2 POD)

- **Only PE0 will write-back!**









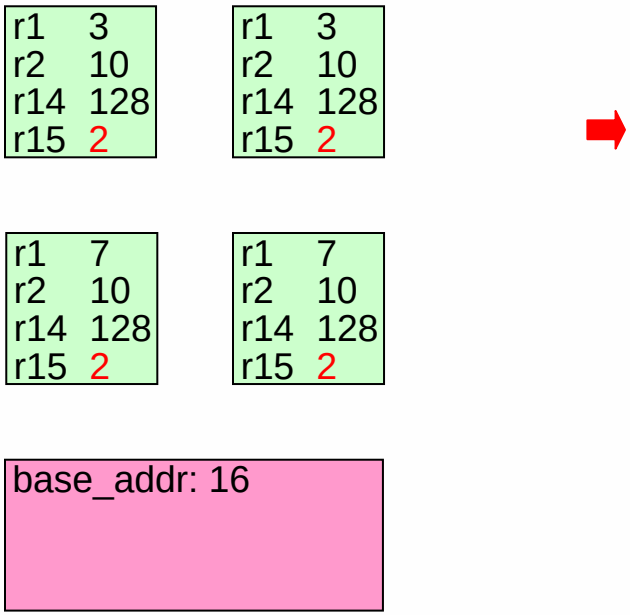
16: 1
17: 2
18: 3
19: 4

128: <- addr of ‘result’

```
int result;

POD_movl( r14, &result );

$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]

$ASM sub r15 = r15, 0
$ASM pushmask.e
        $ASM st sys[ r14 + 0 ] = r15
$ASM popmask

POD_mfence();
```

Programming Model (2x2 POD)

- **Load PE_ID**




```
int result;

POD_movl( r14, &result );

$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]

$ASM sub r15 = r15, 0
$ASM pushmask.e
        $ASM st sys[ r14 + 0 ] = r15
$ASM popmask

POD_mfence();
```

Programming Model (2x2 POD)

- **Compare PE_ID**

r1	3
r2	10
r14	128
r15	2

r1	3
r2	10
r14	128
r15	3

r1	7
r2	10
r14	128
r15	0

r1	7
r2	10
r14	128
r15	1

base_addr: 16

16: 1
17: 2
18: 3
19: 4

128: <- addr of ‘result’

```
int result;

POD_movl( r14, &result );

$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]

$ASM sub r15 = r15, 0
$ASM pushmask.e
        $ASM st sys[ r14 + 0 ] = r15
$ASM popmask

POD_mfence();
```

Programming Model (2x2 POD)

- **Enable PE only when equal to zero**

r1	3
r2	10
r14	128
r15	2

r1	3
r2	10
r14	128
r15	3

r1	7
r2	10
r14	128
r15	0

r1	7
r2	10
r14	128
r15	1

base_addr: 16

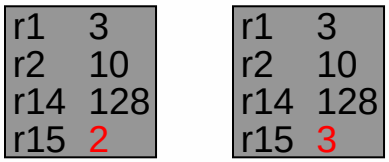
16: 1
17: 2
18: 3
19: 4

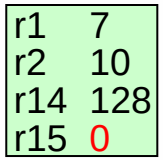
128: <- addr of ‘result’

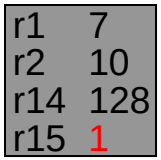
```
int result;

POD_movl( r14, &result );

$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]

$ASM sub r15 = r15, 0
$ASM pushmask.e
        $ASM st sys[ r14 + 0 ] = r15
$ASM popmask

POD_mfence();
```

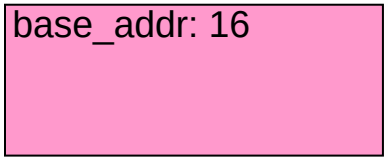
Programming Model (2x2 POD)

- **Write-back**









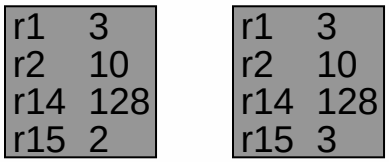
16: 1
17: 2
18: 3
19: 4

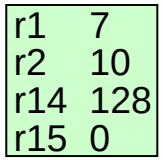
128: <- addr of ‘result’

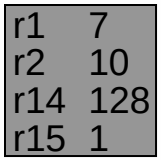
```
int result;

POD_movl( r14, &result );

$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]

$ASM sub r15 = r15, 0
$ASM pushmask.e
        $ASM st sys[ r14 + 0 ] = r15
$ASM popmask

POD_mfence();
```

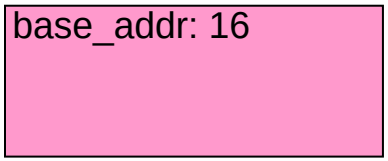
Programming Model (2x2 POD)

- **Enable ALL PEs**









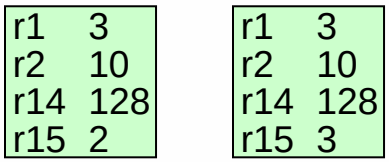
16: 1
17: 2
18: 3
19: 4

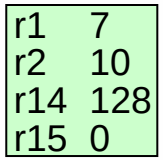
128: <- addr of ‘result’

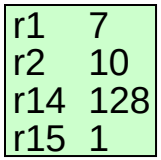
```
int result;

POD_movl( r14, &result );

$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]

$ASM sub r15 = r15, 0
$ASM pushmask.e
        $ASM st sys[ r14 + 0 ] = r15
$ASM popmask

POD_mfence();
```

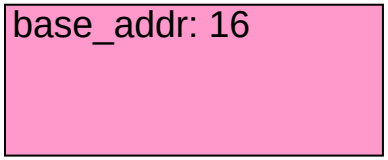
Programming Model (2x2 POD)

- **Wait until the result is written-back**









16: 1
17: 2
18: 3
19: 4

128: <- addr of ‘result’

```
int result;

POD_movl( r14, &result );

$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]

$ASM sub r15 = r15, 0
$ASM pushmask.e
        $ASM st sys[ r14 + 0 ] = r15
$ASM popmask

POD_mfence();
```