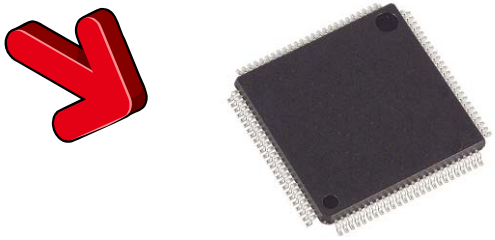


Many-cores: Supercomputer-on-chip
How many? And how?
(how not to?)

Ran Ginosar

Technion

Feb 2009

Disclosure and Ack

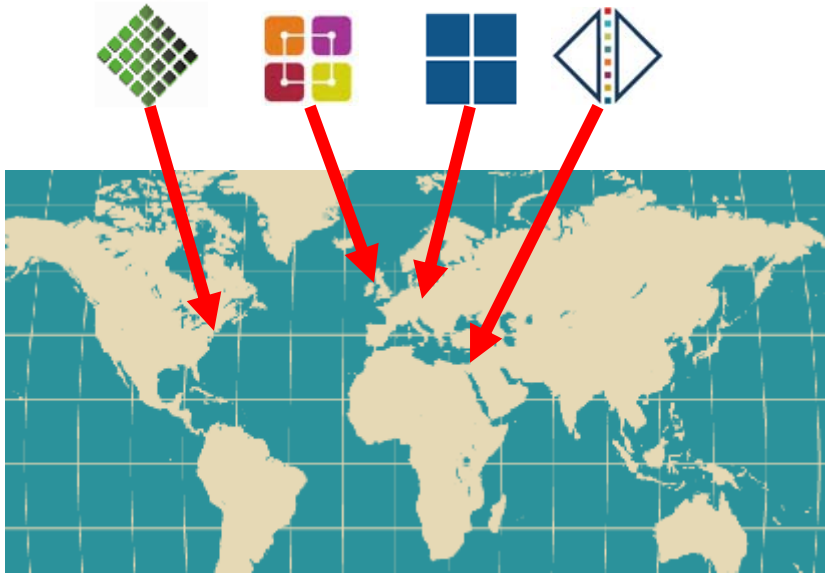
- I am co-inventor / co-founder of Plurality
  - Based on 30 years of (on/off) research
- Presentation ideas stolen freely from others
  - Suddenly there are many experts at and around the Technion ☺

Many-cores

- CMP / Multi-core is “more of the same”
  - Several high-end complex powerful processors
  - Each processor manages itself
  - Each processor can execute the OS
  - Good for many unrelated tasks (e.g. Windows)
  - Reasonable on 2–8 processors, then it breaks
- Many-cores
  - 100 – 1,000 – 10,000
  - Useful for heavy compute-bound tasks
  - So far (50 years) many disasters
    - But there is light at the end of the tunnel ☺

Agenda

- Review 4 cases
- Analyze
- How *NOT* to make a many-core



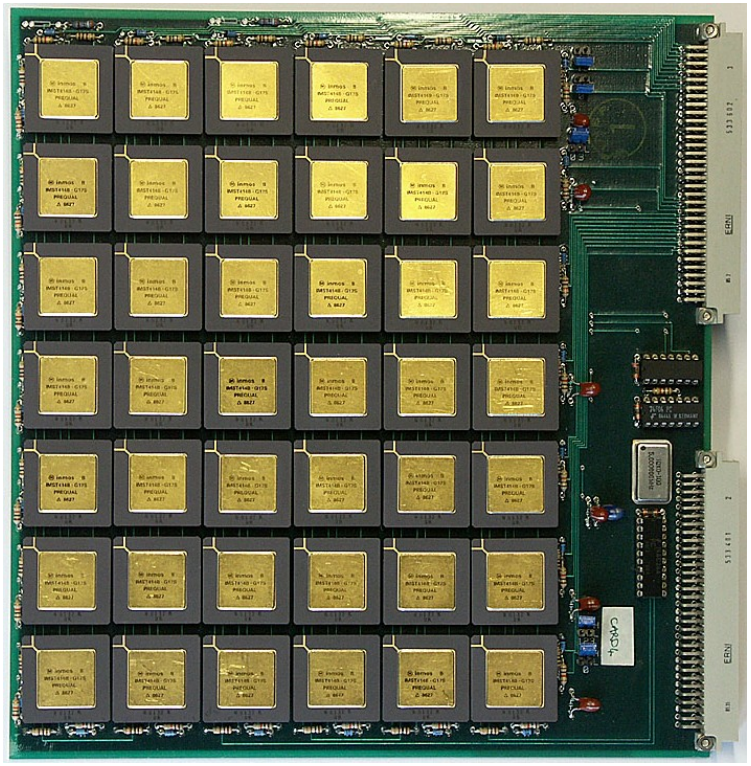
Many many-core contenders

- Ambric
- Aspex Semiconductor
- ATI GPGPU
- BrightScale
- ClearSpeed Technologies
- Coherent Logix, Inc.
- CPU Technology, Inc.
- Element CXI
- Elixent/Panasonic
- IBM Cell
- IMEC
- Intel Larrabee
- Intellasys
- IP Flex
- MathStar
- Motorola Labs
- NEC
- Nvidia GPGPU
- PACT XPP
- Picochip
- Plurality
- Rapport Inc.
- Recore
- Silicon Hive
- Stream Processors Inc.
- Tabula
- Tilera

(many are dead / dying / will die / should die)

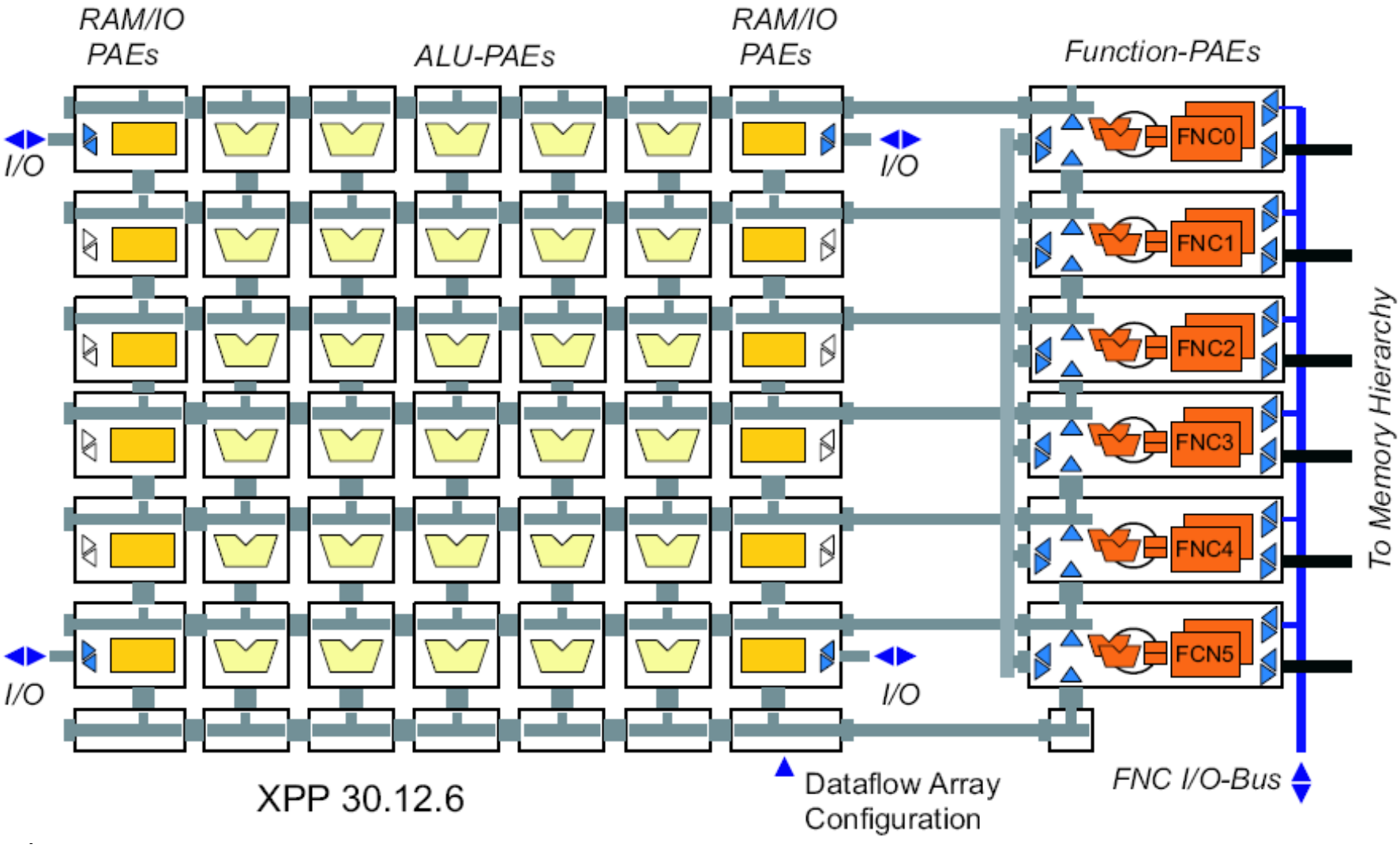
PACT XPP

- German company, since 1999
  - Martin Vorbach,
    an ex-user of Transputers

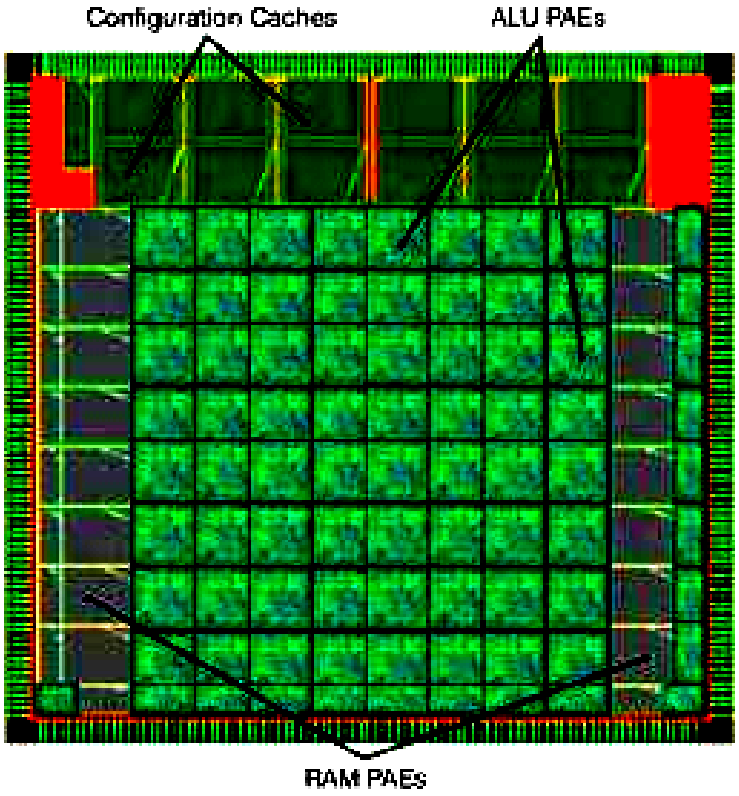


42x Transputers mesh 1980s

PACT XPP (96 elements)



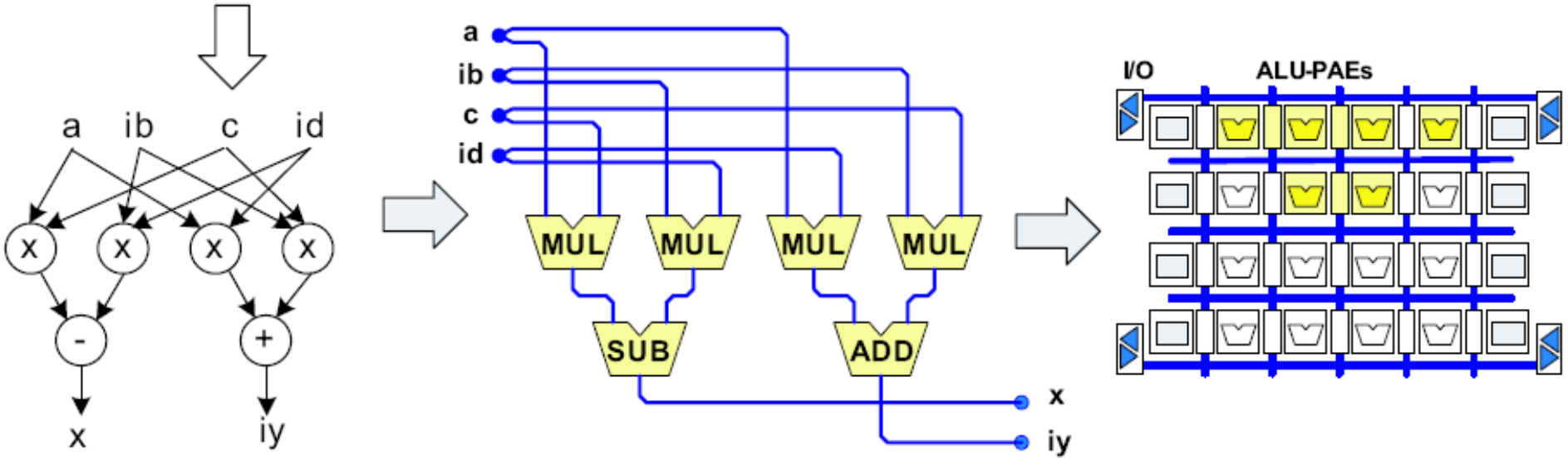
PACT XPP die photo



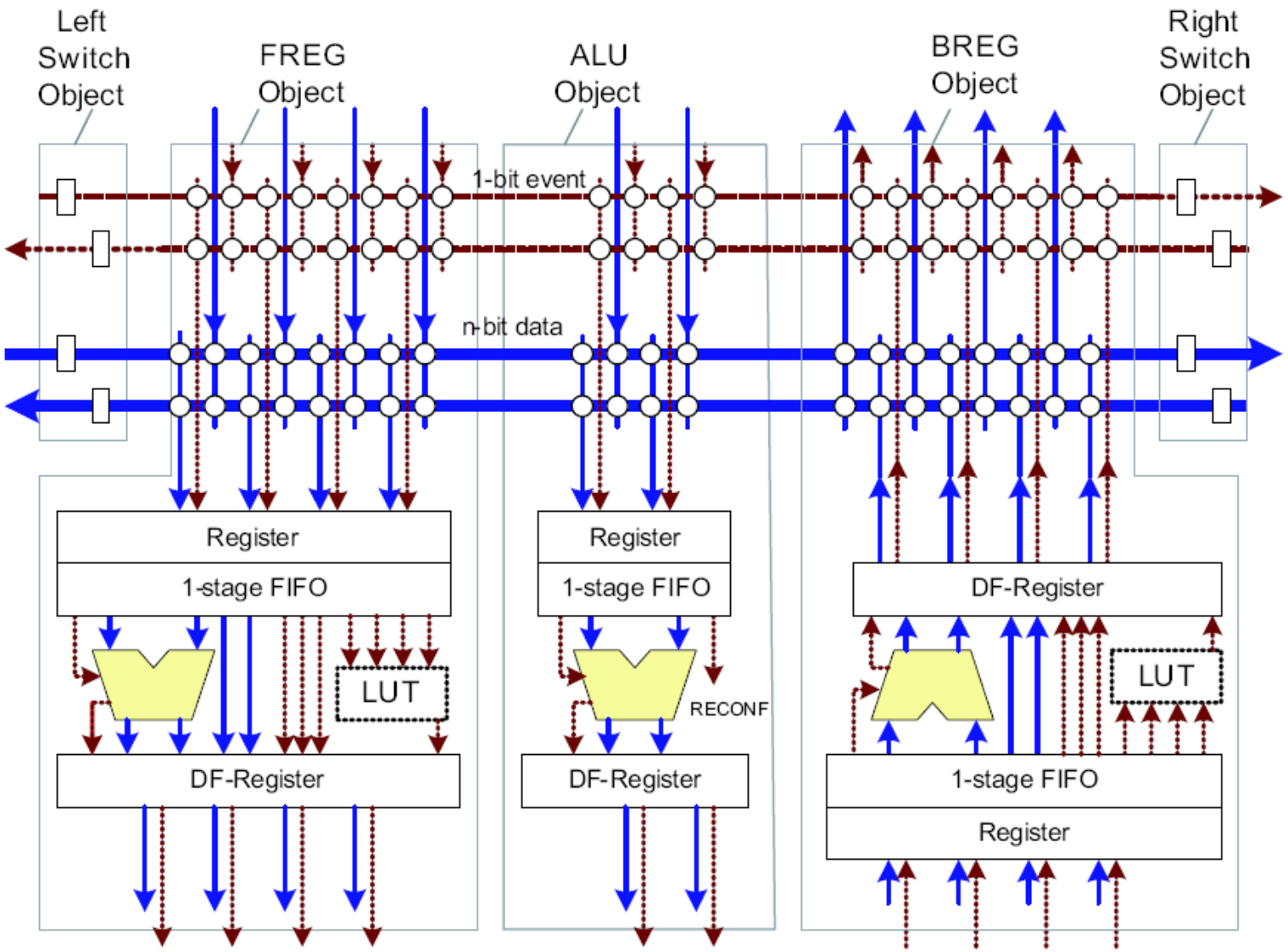
PACT: Static mapping, circuit-switch reconfigured NoC

PACT

x + iy = (a+ib) * (c+id)
= (ac - bd) + i (ad + bc)

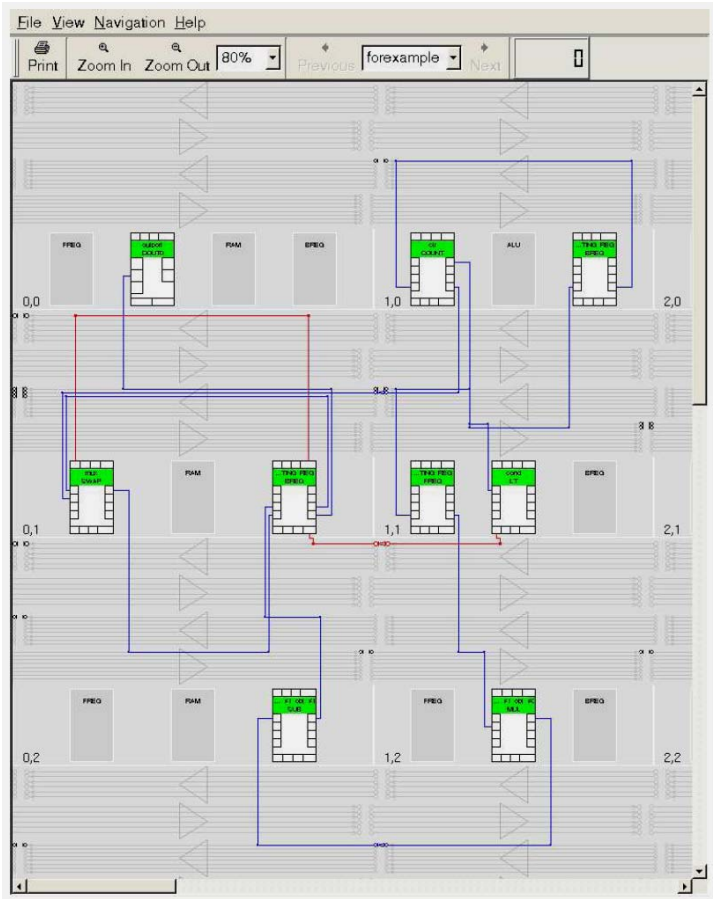


PACT ALU-PAE


Left
Switch
Object
FREG
Object
ALU
Object
BREG
Object
Right
Switch
Object
1-bit event
n-bit data
Register
1-stage FIFO
LUT
DF-Register
Register
1-stage FIFO
RECONF
DF-Register
DF-Register
LUT
1-stage FIFO
Register

PACT

- Static task mapping ☹
  - And a debug tool for that

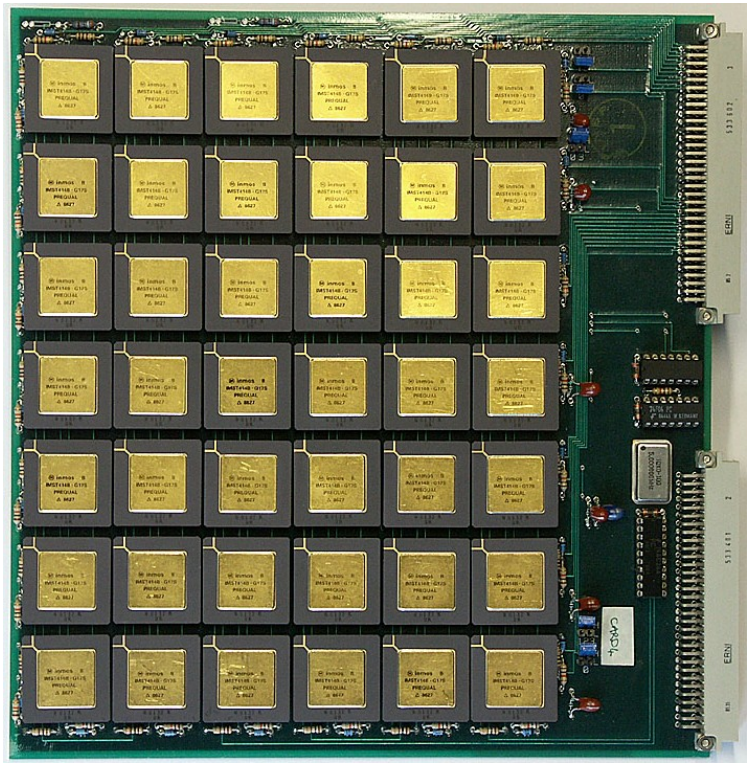


PACT analysis

- Fine granularity computing ☺
- Heterogeneous processors ☹
- Static mapping
  → complex programming ☹
- Circuit-switched NoC → static reconfigurations
  → complex programming ☹
- Limited parallelism
- Doesn’t scale easily

picoChip

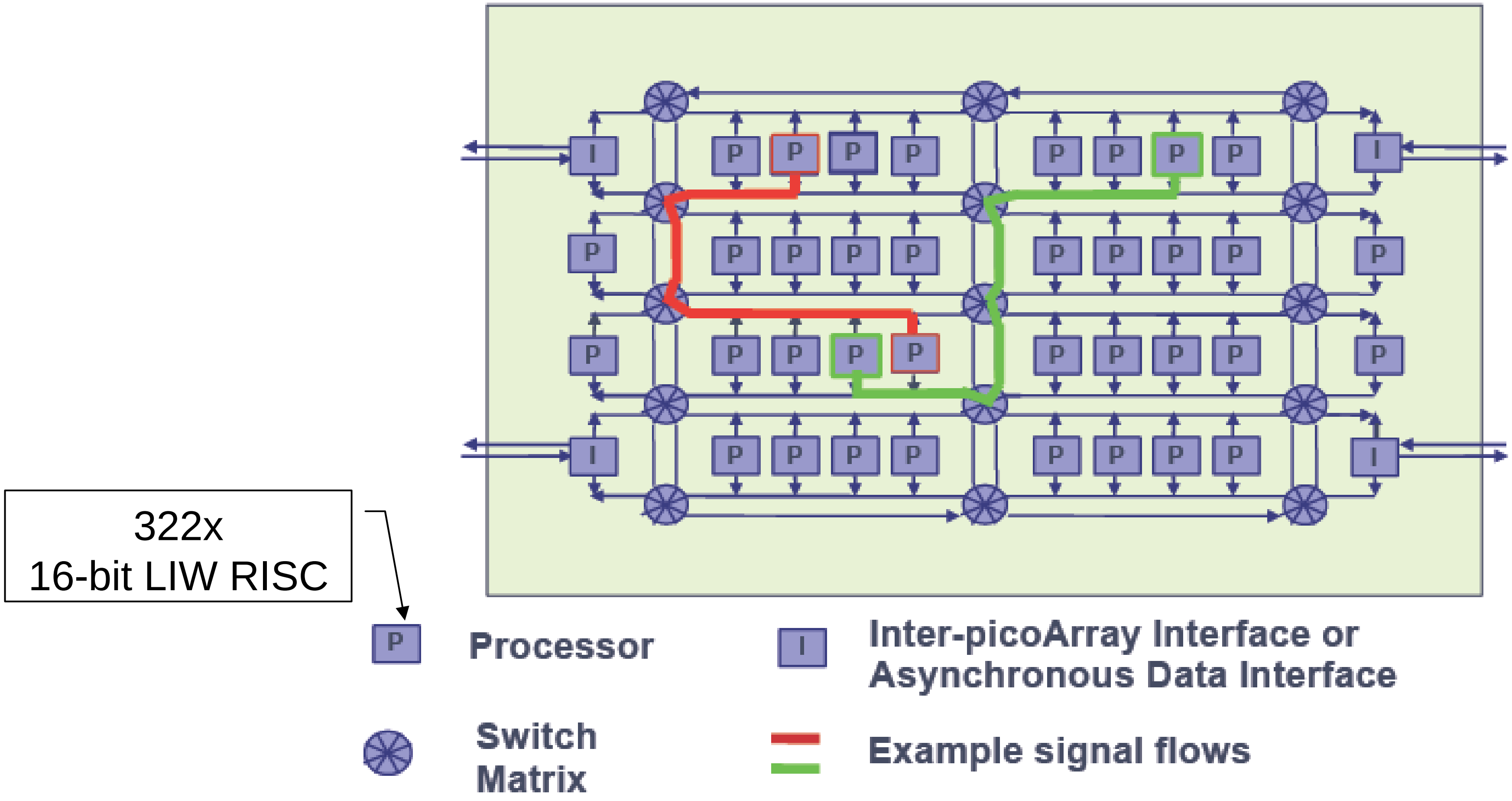
- UK company
- Inspired by Transputers (1980s), David May



42x Transputers mesh 1980s

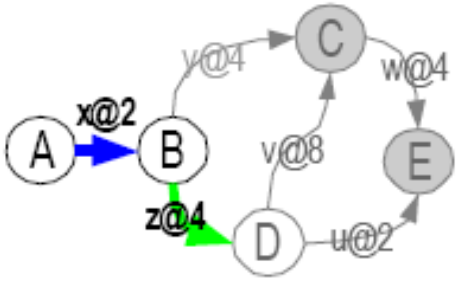
picoChip

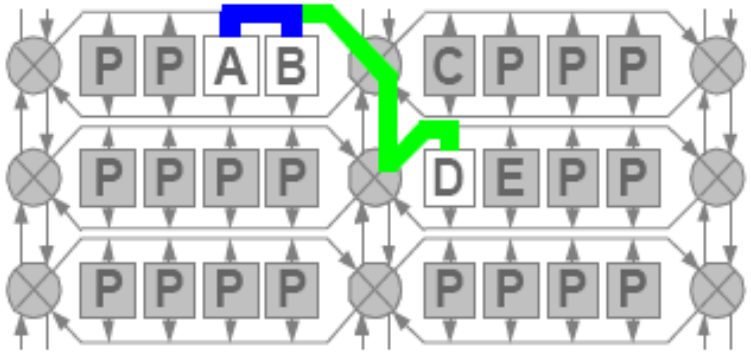
The picoArray concept : Architecture overview



picoChip

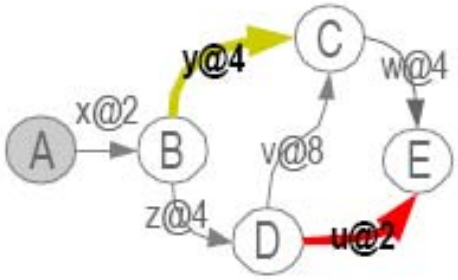
The picoArray concept : picoBus

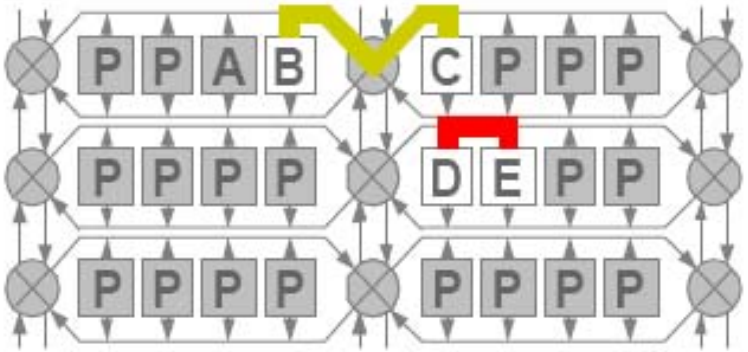




	0	1	2	3	4	5	6	7
U								
V								
W								
X								
Y								
Z								

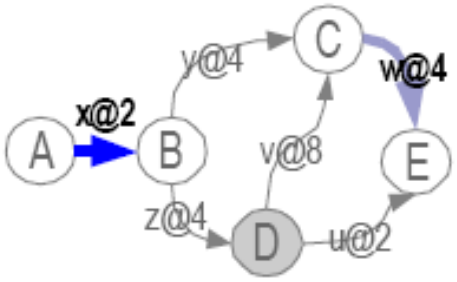
picoChip

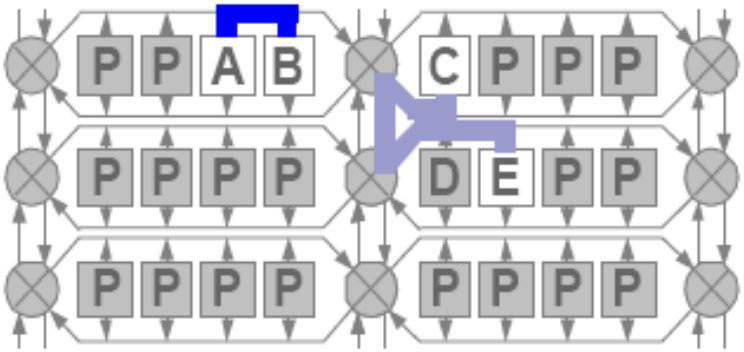




	0	1	2	3	4	5	6	7
U								
V								
W								
X								
Y								
Z								

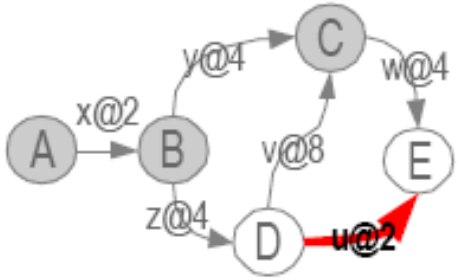
picoChip

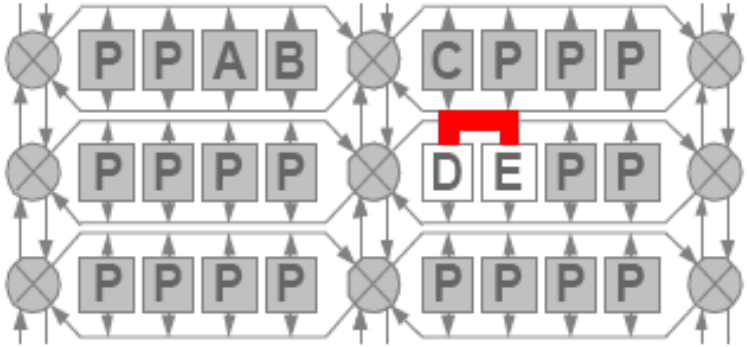




	0	1	2	3	4	5	6	7
U								
V								
W								
X								
Y								
Z								

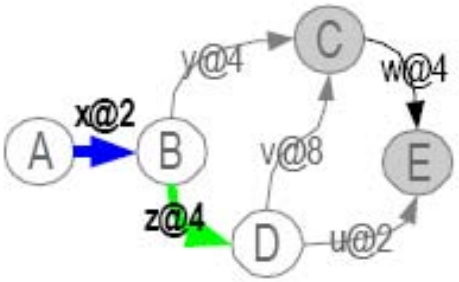
picoChip

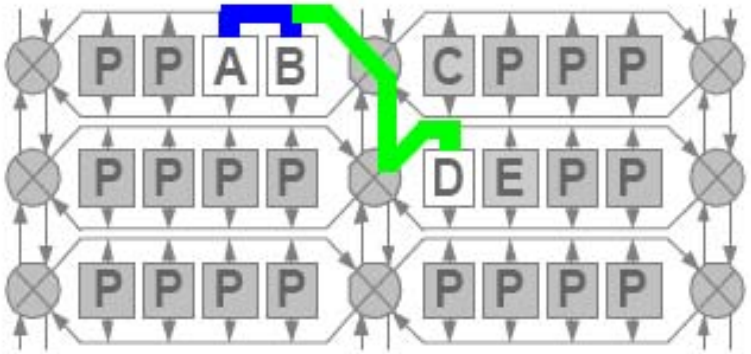




	0	1	2	3	4	5	6	7
U								
V								
W								
X								
Y								
Z								

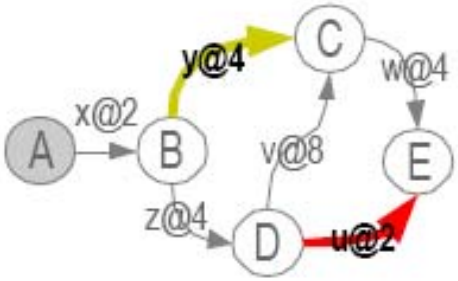
picoChip

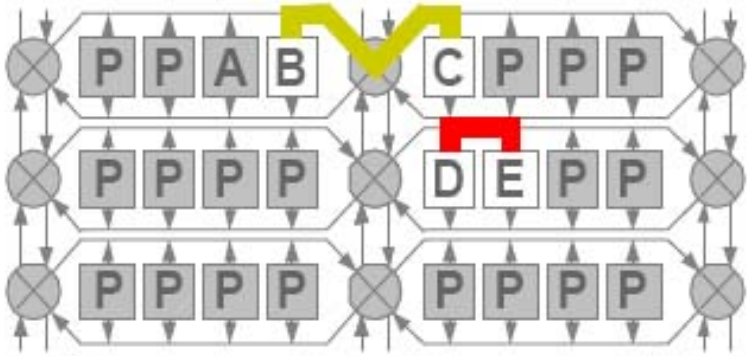


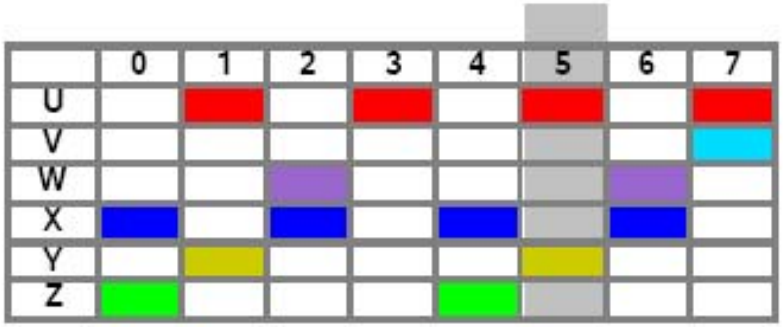


	0	1	2	3	4	5	6	7
U								
V								
W								
X								
Y								
Z								

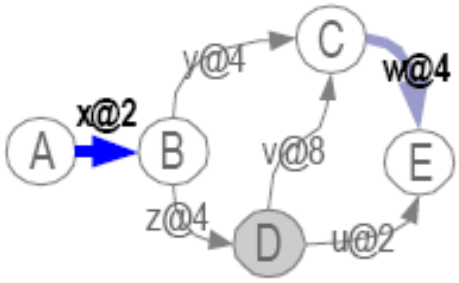
picoChip

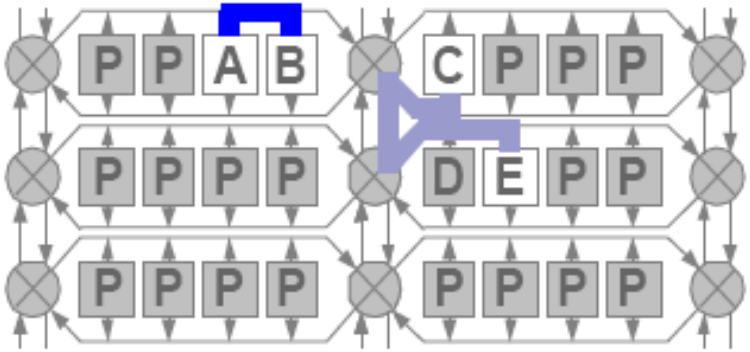

A
B
C
D
E
x@2
y@4
w@4
v@8
z@4
u@2


P P A B C P P P
P P P P D E P P
P P P P P P P P


0 1 2 3 4 5 6 7
U
V
W
X
Y
Z

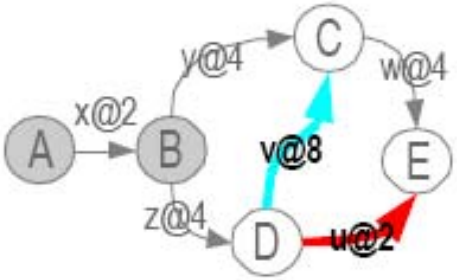
picoChip

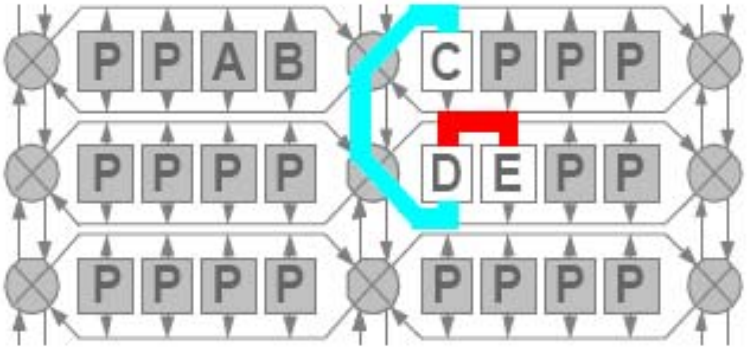




	0	1	2	3	4	5	6	7
U								
V								
W								
X								
Y								
Z								

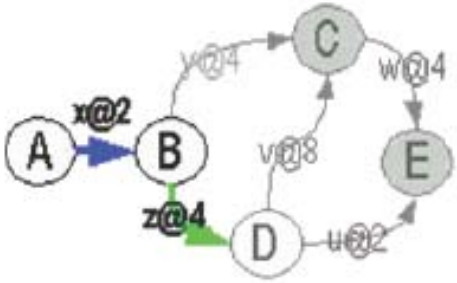
picoChip

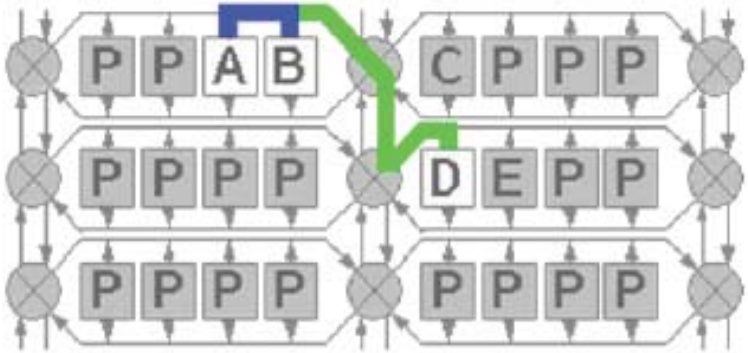




	0	1	2	3	4	5	6	7
U								
V								
W								
X								
Y								
Z								

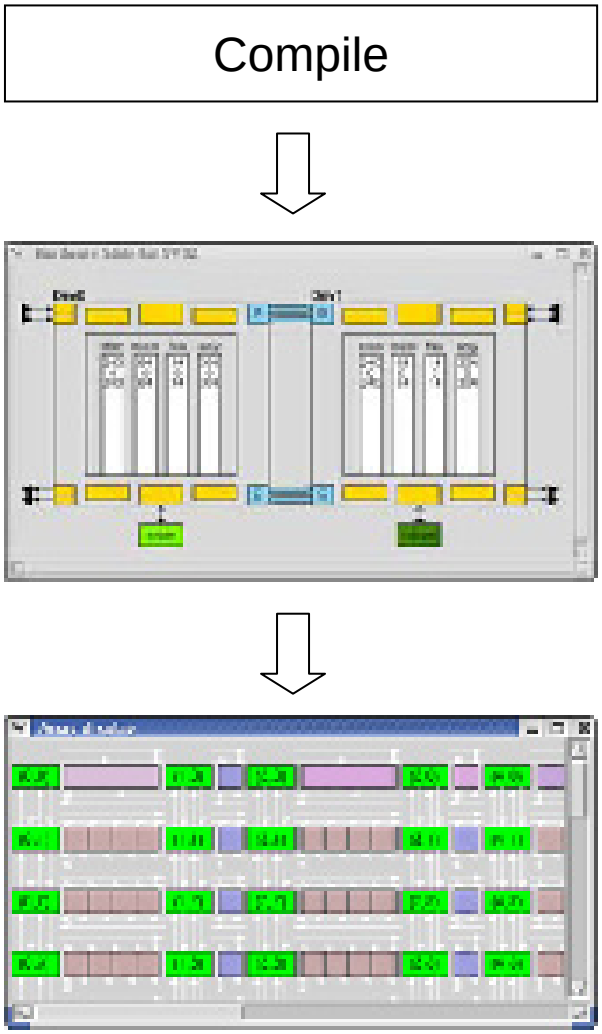
picoChip





	0	1	2	3	4	5	6	7
U								
V								
W								
X								
Y								
Z								

picoChip: Static Task Mapping ☹

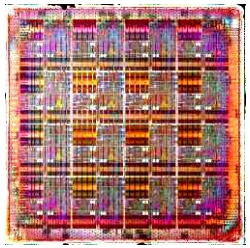


picoChip analysis

- MIMD, fine granularity, homogeneous cores ☺
- Static mapping
  - → complex programming ☹
- Circuit-switched NoC → static reconfigurations
  - → complex programming ☹
- Doesn't scale easily
  - Can we create / debug / understand static mapping on 10K?

TILERA®

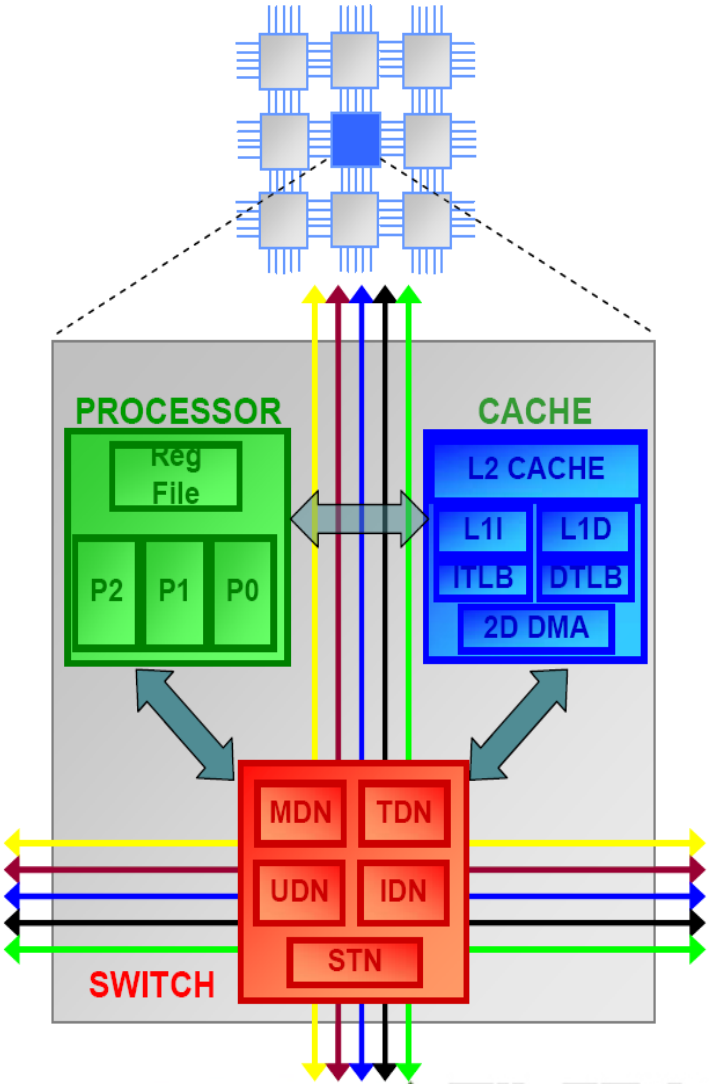
- USA company
- Based on RAW research @ MIT (A. Agarwal)




  - Heavy DARPA funding, university IP
- Classic homogeneous MIMD on mesh NoC
  - “Upgraded” Transputers with “powerful” uniprocessor features
    - Caches ☹
    - Complex communications ☹
- “tiles era”

TILERA Tiles

- Powerful processor
- High freq: ~1 GHz
  - High power (0.5W) ☹
- 5-mesh NoC
  - P-M / P-P / P-IO
- 2.5 levels cache ☹☹
  - L1+ L2
  - Can fetch from L2 of others
- Variable access time
  - 1 – 7 – 70 cycles

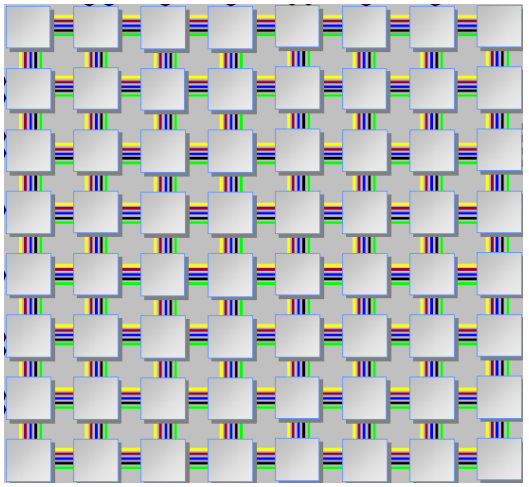


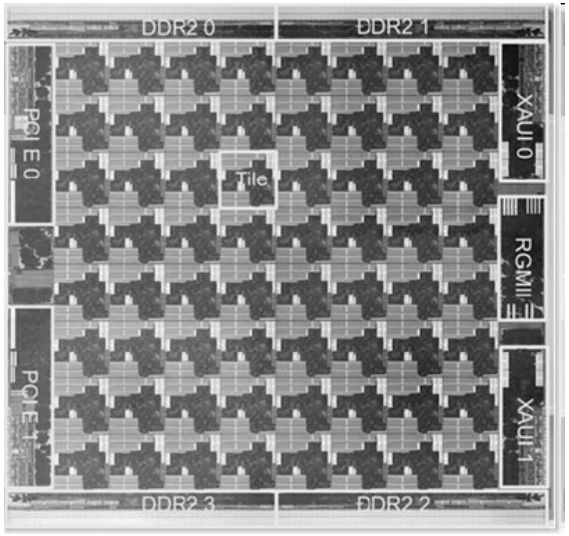
Caches Kill Performance

- Cache is great for a single processor
  - Exploits locality (in time and space)
- Locality only happens locally on many-cores
  - Other (shared) data are buried elsewhere
- Caches help speed up parallel (local) phases
  - Amdahl [1967]: the challenge is NOT the parallel phases

TILERA Array

- 36-64 processors
  - MIMD / SIMD ☹
- Total 5+ MB memory
  - In distributed caches
- High power
  - ~27W ☹☹

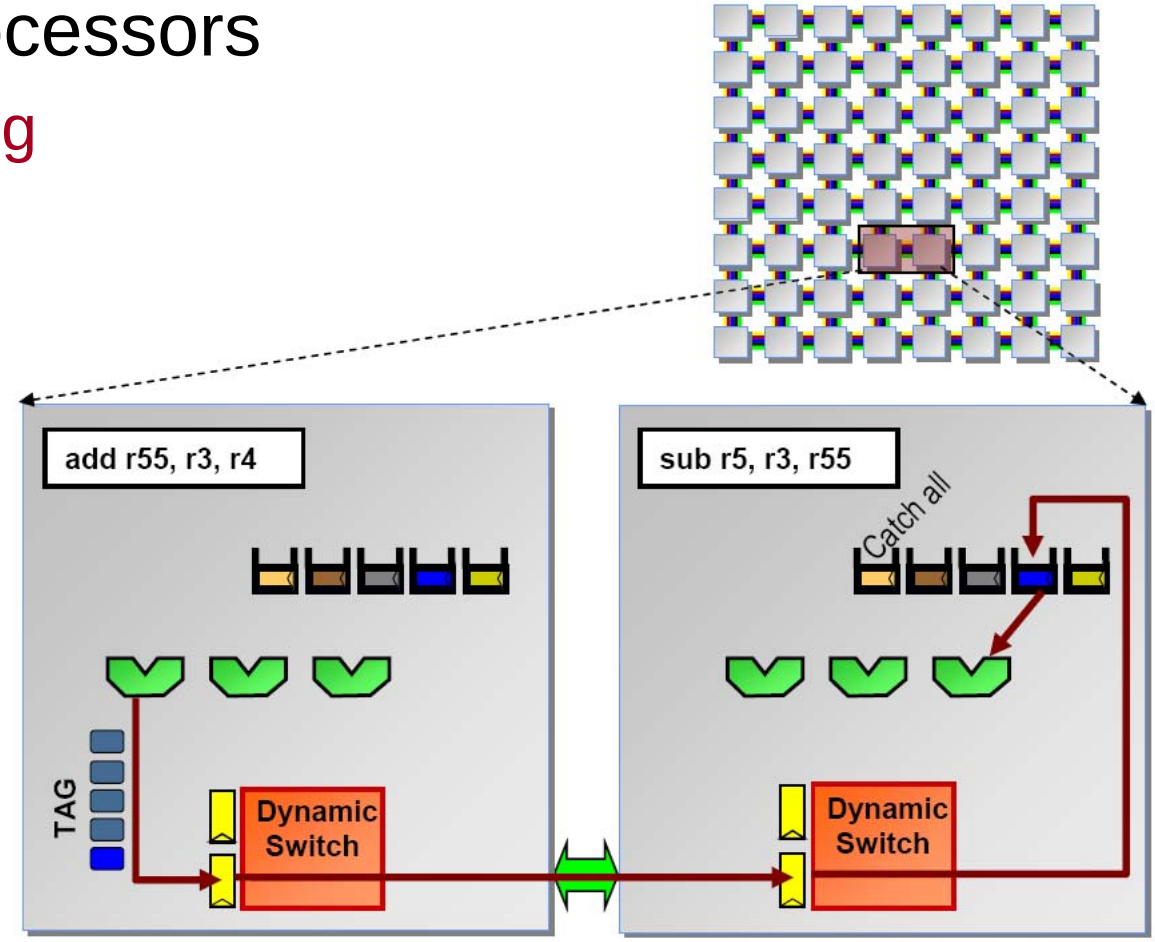




Die photo

TILERA allows statics

- Pre-programmed streams span multi-processors
  - Static mapping

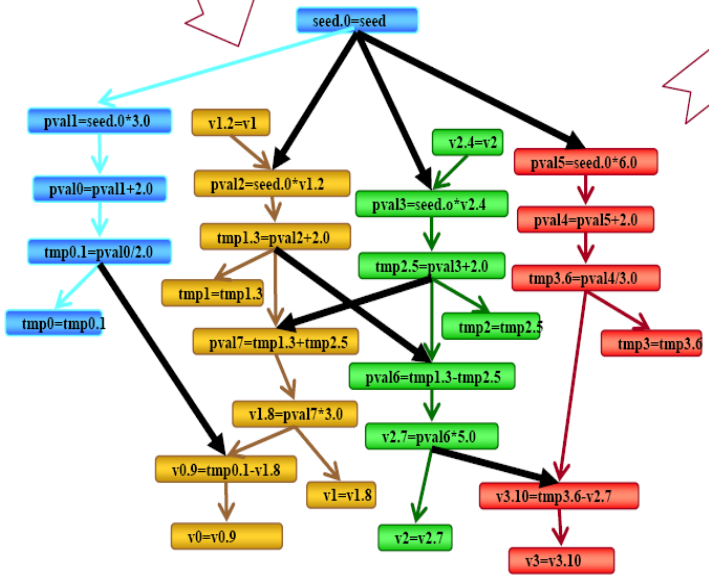


TILERA co-mapping: code, memory, routing ☹

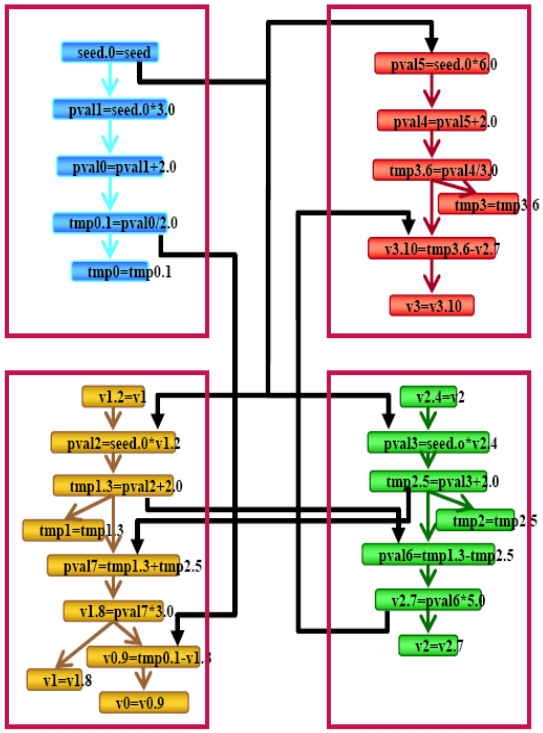
```
tmp0 = (seed*3+2)/2
tmp1 = seed*v1+2
tmp2 = seed*v2 + 2
tmp3 = (seed*6+2)/3
v2 = (tmp1 - tmp3)*5
v1 = (tmp1 + tmp2)*3
v0 = tmp0 - v1
v3 = tmp3 - v2
```

C

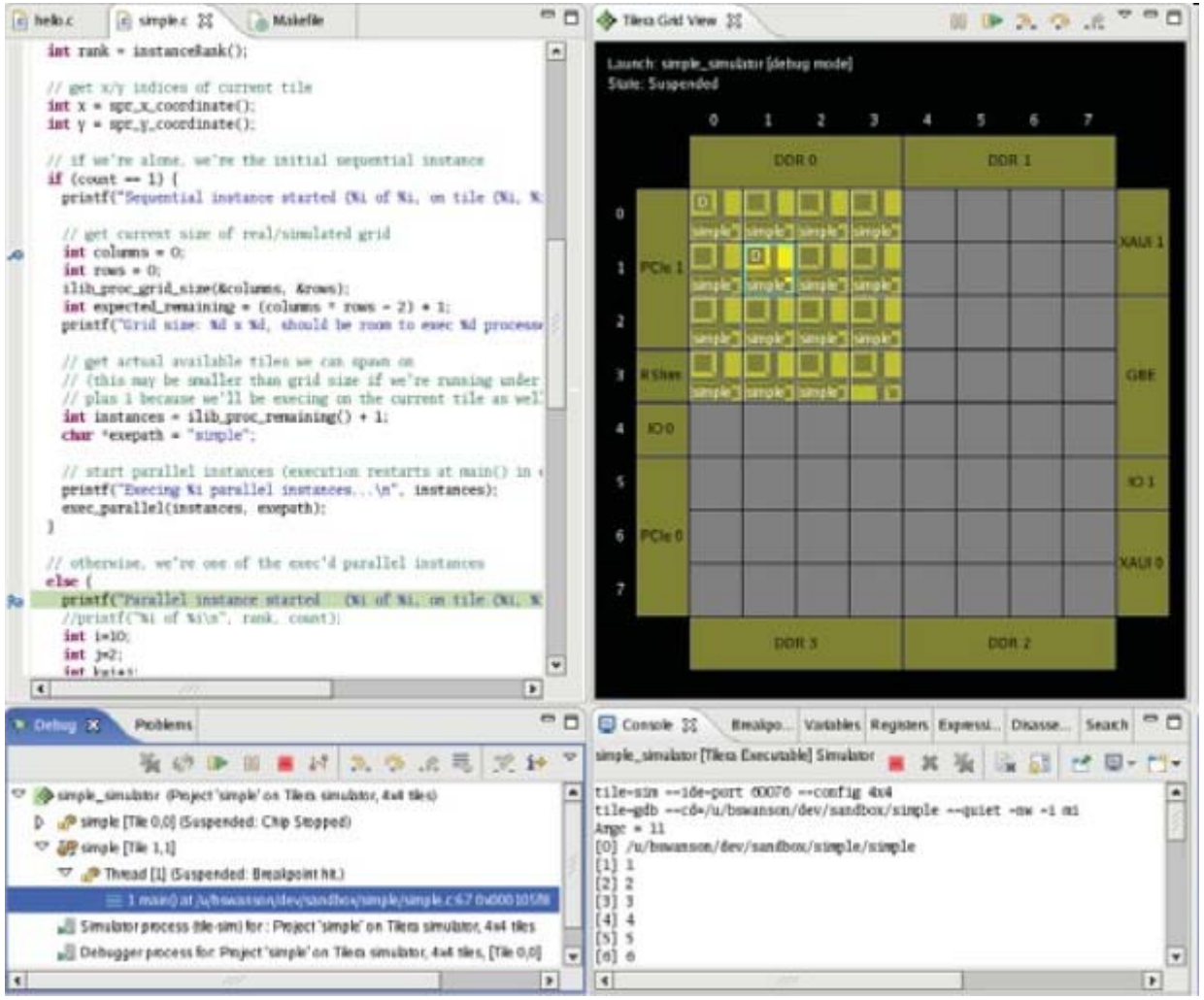
Partitioning



Place, Route, Schedule



TILERA static mapping debugger ☹



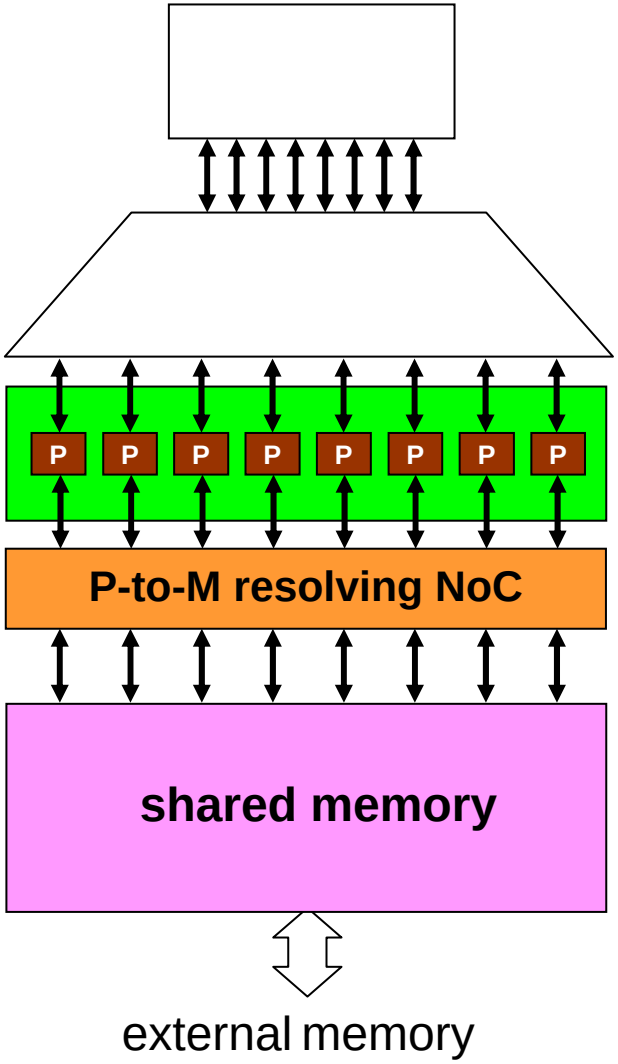
TILERA® analysis

- Achieves good performance
- Bad on power
- Hard to scale
- Hard to program

PLURALITY

- Israel
- Technion research (since 1980s)

PLURALITY Architecture: Part I

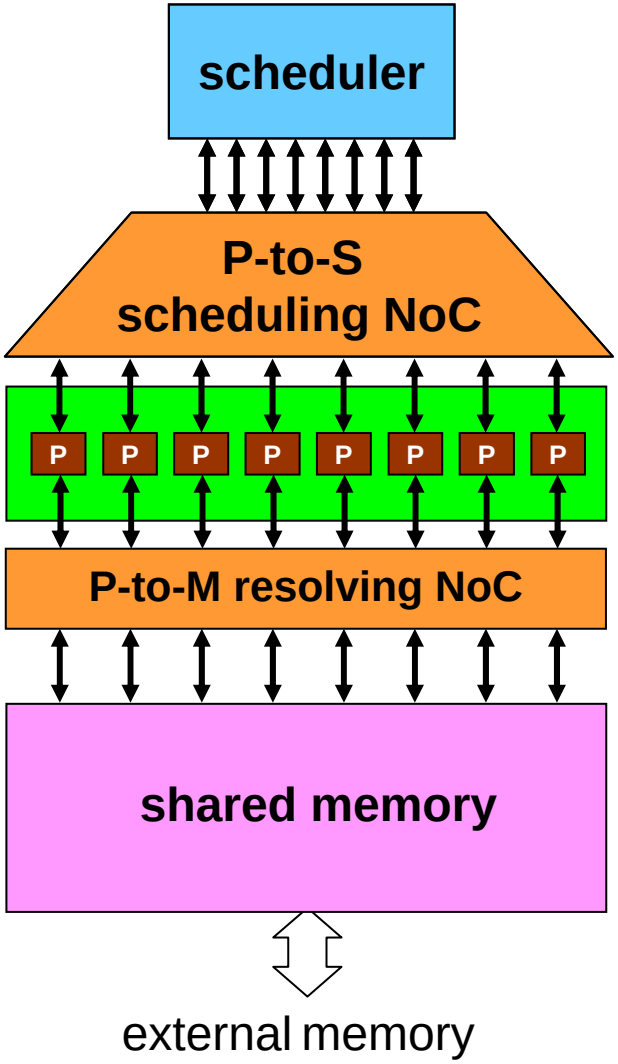


fine granularity
NO PRIVATE MEMORY

tightly coupled memory
equi-distant (1 cycle each way)
fast combinational NOC

“anti-local” addressing by interleaving
MANY banks / ports
negligible conflicts

PLURALITY Architecture: Part II



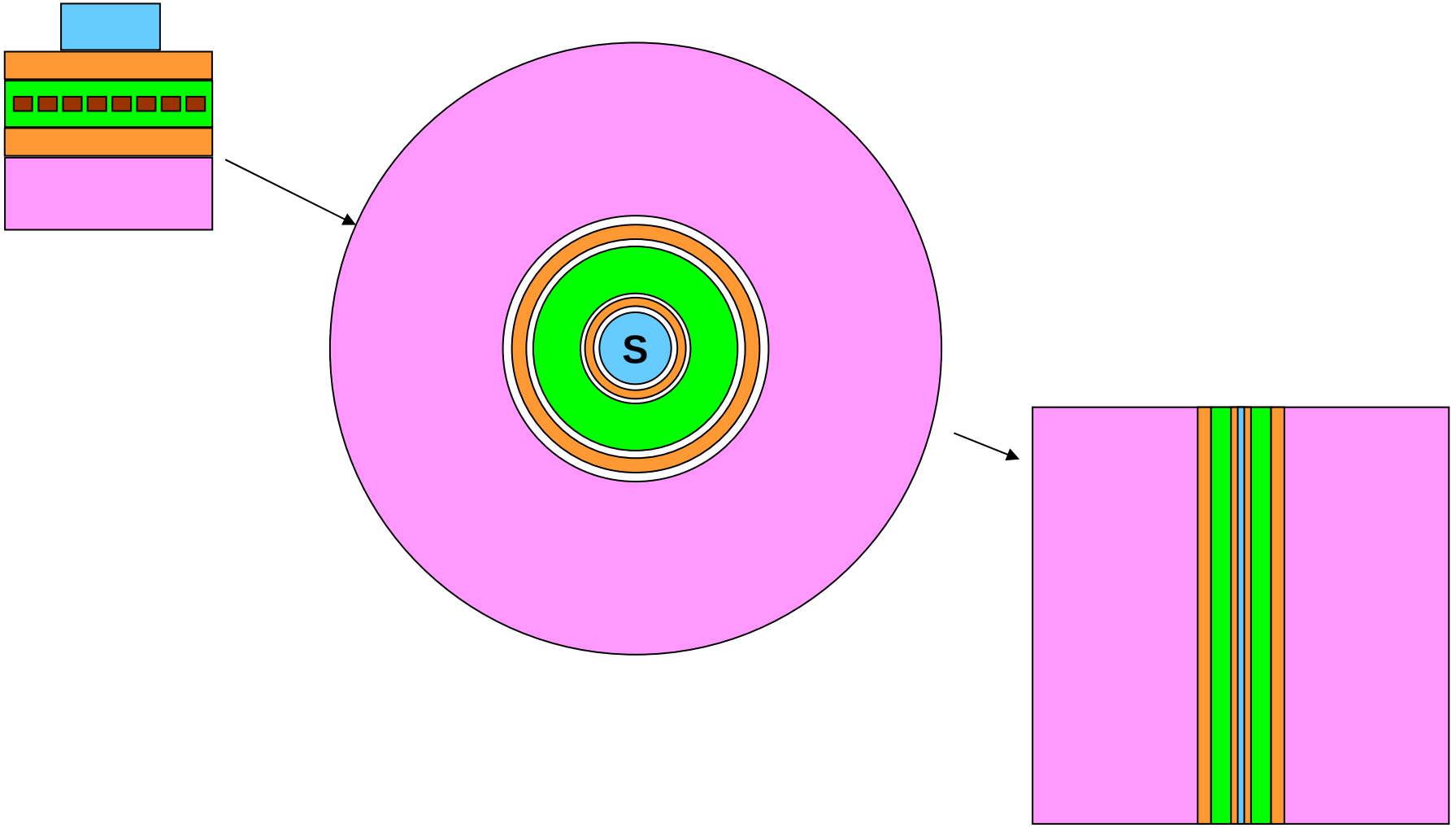
low latency parallel scheduling
enables fine granularity

fine granularity
NO PRIVATE MEMORY

tightly coupled memory
equi-distant (1 cycle each way)
fast combinational NOC

“anti-local” addressing by interleaving
MANY banks / ports
negligible conflicts

PLURALITY Floorplan

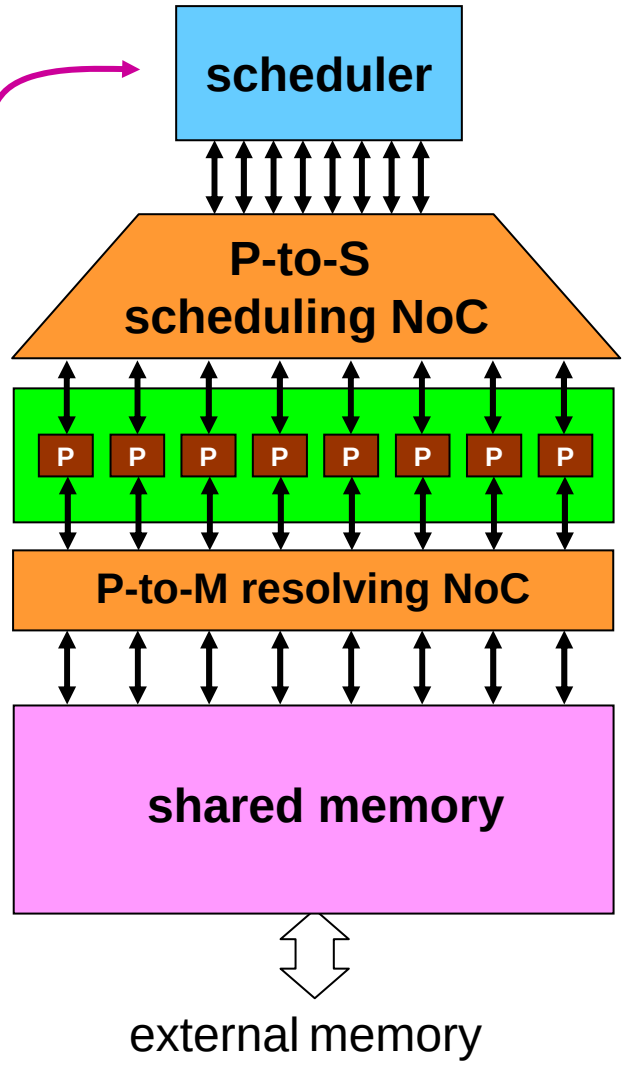


PLURALITY programming model

- Compile into
  - task-dependency-graph = ‘task map’
  - task codes
- Task maps loaded into scheduler
- Tasks loaded into memory

Task template:

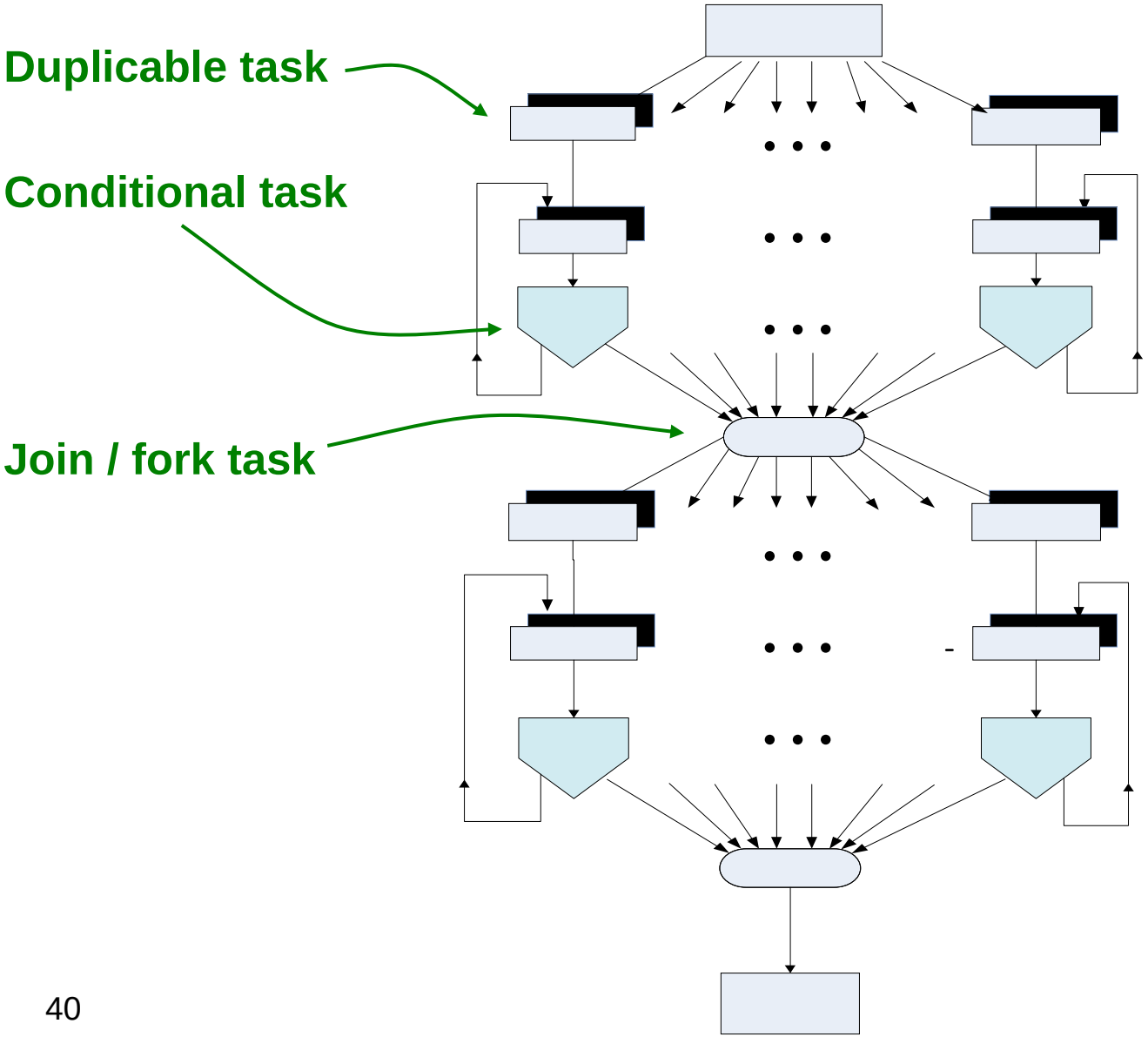
```
regular
duplicable      task xxx( dependencies )
join/fork
{
        … INSTANCE ….
        …..
}
```



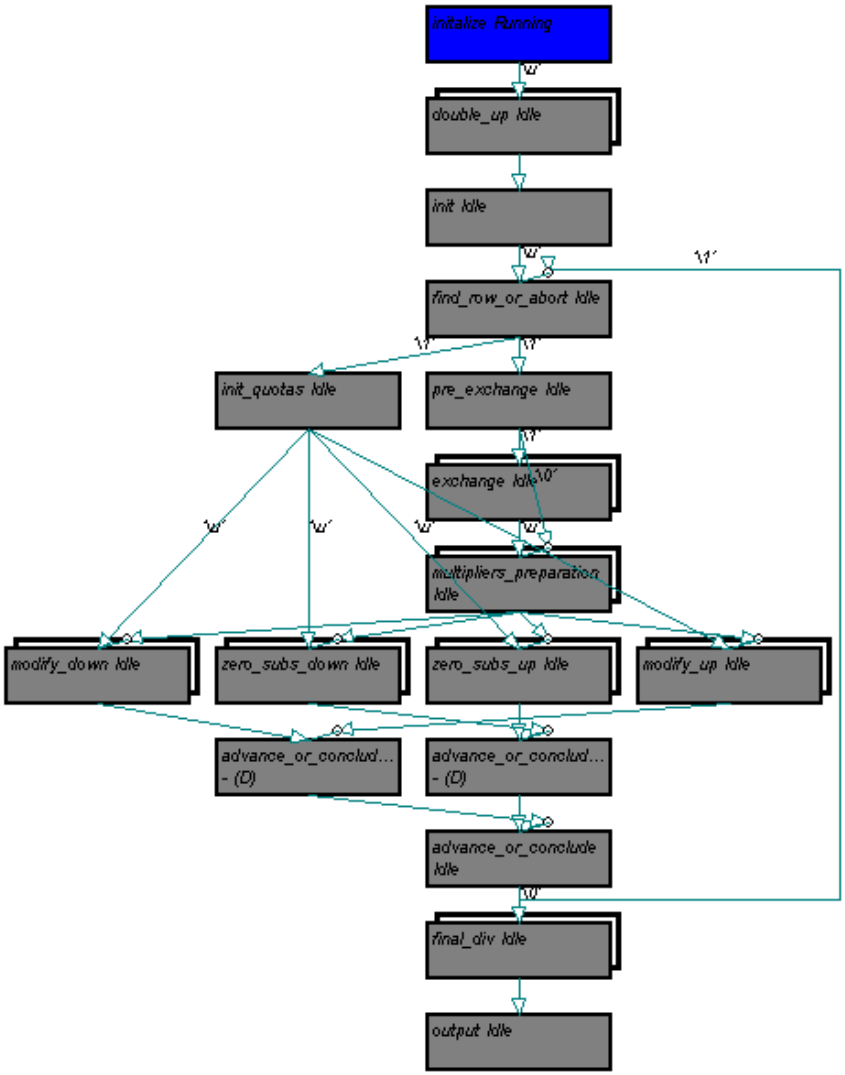
Fine Grain Parallelization

- Convert (independent) loop iterations
  - `for ( i=0; i<10000; i++ ) { a[i] = b[i]*c[i]; }`
- into parallel tasks
  - ```
    duplicable task XX(…) 10000
    {  ii = INSTANCE;
       a[ii] = b[ii]*c[ii];
    }
    ```
- All tasks, or any subset, can be executed in parallel
    ```

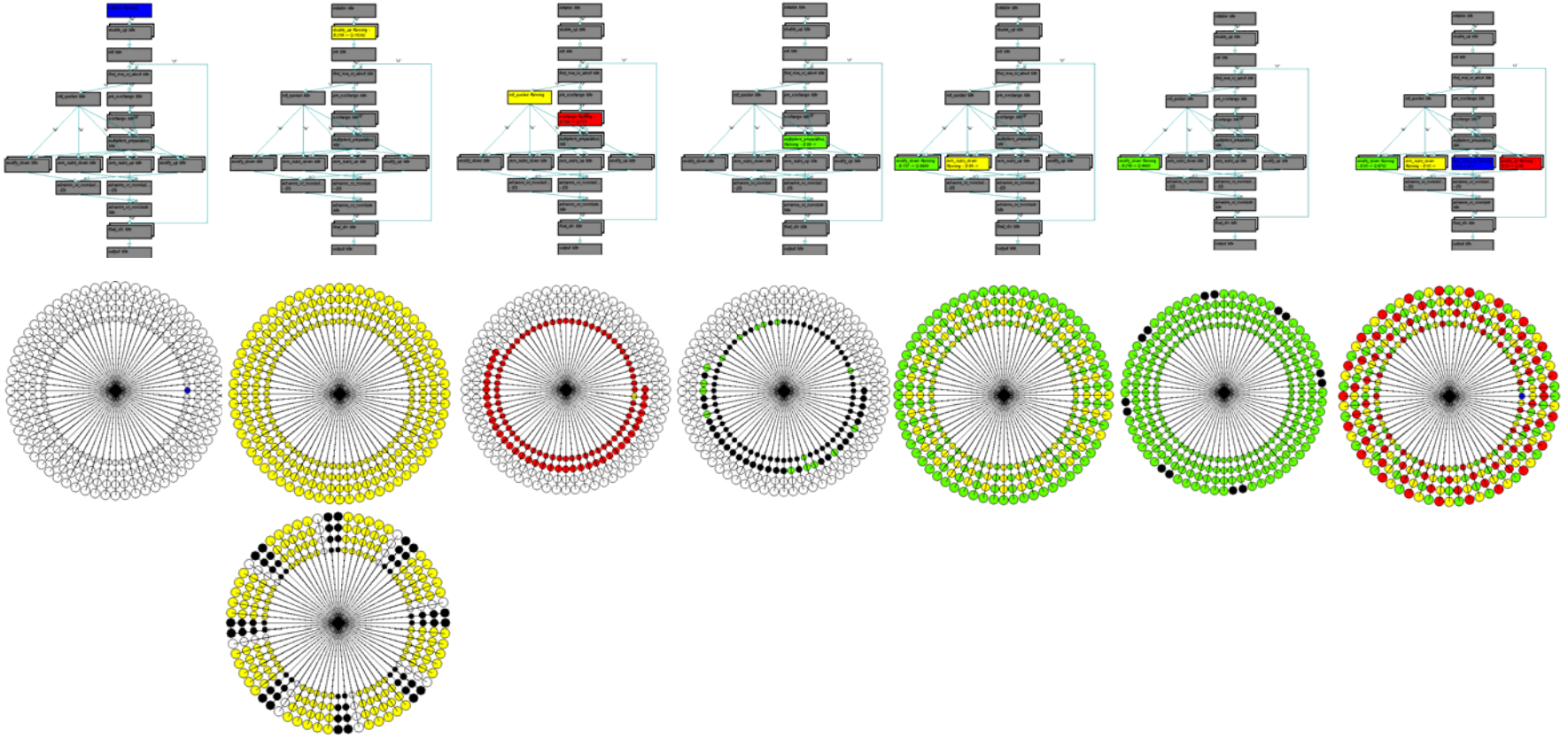
Task map example (2D FFT)



Another task map (linear solver)



Linear Solver: Simulation snap-shots



PLURALITY Architectural Benefits

- Shared, uniform (equi-distant) memory
  - no worry which core does what
  - no advantage to any core because it already holds the data
- Many-bank memory + fast P-to-M NoC
  - low latency
  - no bottleneck accessing shared memory
- Fast scheduling of tasks to free cores (many at once)
  - enables fine grain data parallelism
  - impossible in other architectures due to:
    - task scheduling overhead
    - data locality
- Any core can do any task equally well on short notice
  - scales automatically
- Programming model:
  - intuitive to programmers
  - easy for automatic parallelizing compiler

PLURALITY

- Target design (no silicon yet)
  - 256 cores
  - 500 MHz
    - For 2 MB, slower for 20 MB
  - Access time: 2 cycles (+)
  - 3 Watts
- Designed to be
  - Attractive to programmers (simple)
  - Scalable
  - Fight Amdahl’s rule

Analysis

The VLSI-aware many-core (crude) analysis

	One core	N-core
Area	a	A (fixed)
Num. processors	1	$N = A/a$
Frequency	$f = \sqrt{a}$	$f = \sqrt{a} = \sqrt{\frac{A}{N}}$
Performance	$\sqrt{a}$	$N\sqrt{a} = \sqrt{NA}$
Power	$p = af = a\sqrt{a}$	$P = Np = A\sqrt{a} = \frac{A\sqrt{A}}{\sqrt{N}}$
Perf/Power		$\propto N$

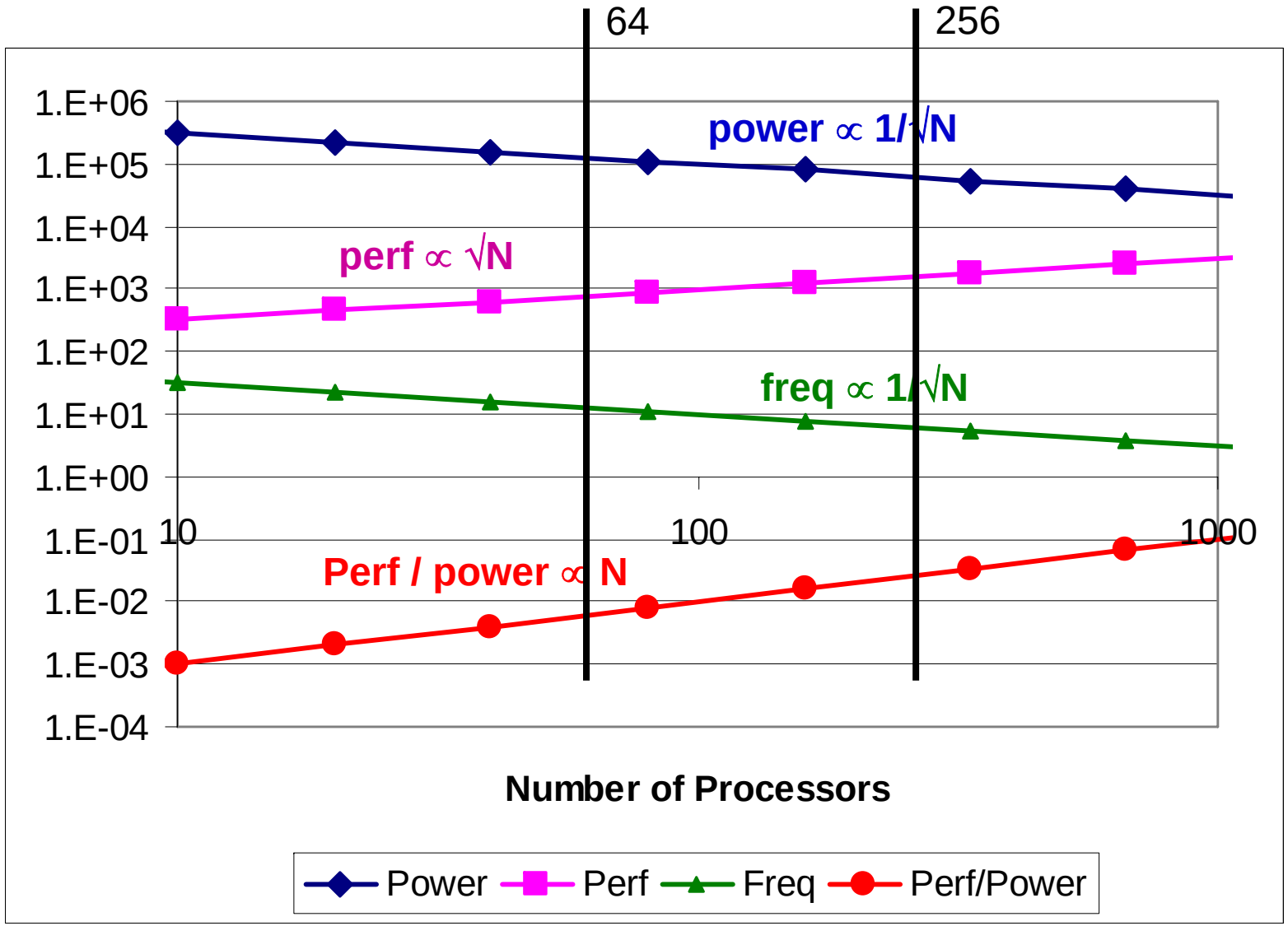
Common error **I**:
Assume that a is fixed

Common error **II**:
Maximize frequency

Common error **III**:
Assume performance is linear in N

Common error **IV**:
Assume power is linear in N

The VLSI-aware many-core (crude) analysis



things we shouldn’t do in many-cores

- No processor-sensitive code
  - No heterogeneous processors
- No speculation
  - No speculative execution
  - No speculative storage (aka cache)
  - No speculative latency (aka packet-switched or circuit-switched NoC)
- No bottlenecks
  - No scheduling bottleneck (aka OS)
  - No issue bottlenecks (aka multithreading)
  - No memory bottlenecks (aka local storage)
- No programming bottlenecks
  - No multithreading / GPGPU / SIMD / static mappings / heterogeneous processors / …
- No statics
  - No static task mapping
  - No static communication patterns

Conclusions

- Powerful processors are inefficient
- Principles of high-end CPU are damaging
  - Speculative anything, cache, locality, hierarchy
- Complexity harms (when exposed)
  - Hard to program
  - Doesn’t scale
- Hacking (static anything) is hacking
  - Hard to program
  - Doesn’t scale
- Keep it simple, stupid [Pythagoras, 520 BC]