The future is parallel but it may not be easy

Michael J. Flynn

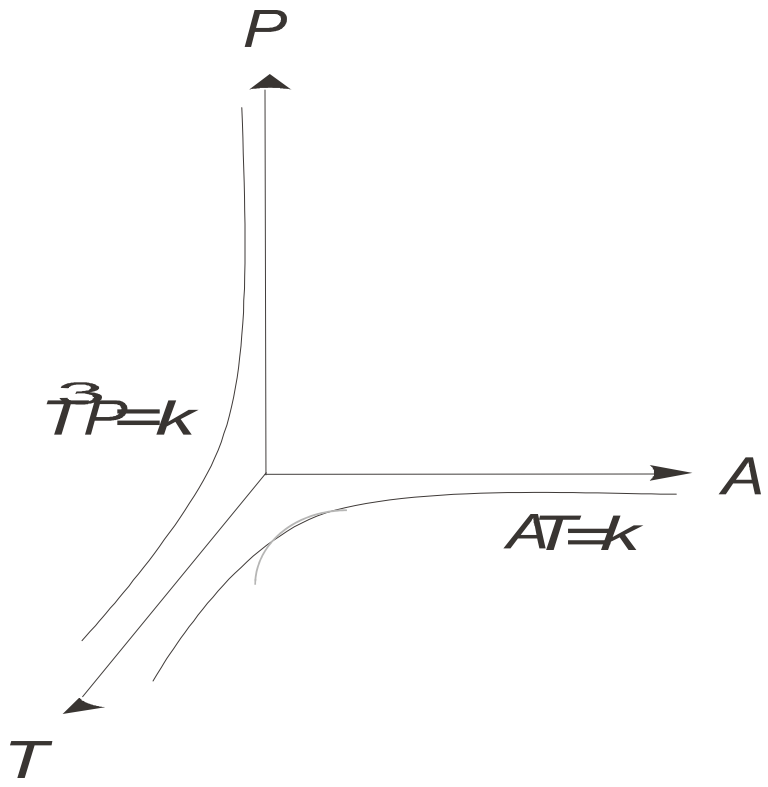
Maxeler and Stanford University

Outline I

- The big technology tradeoffs: *area, time, power*
- HPC: What's new at the parallel processor node
- H^2PC: And beyond the cluster / node for really big applications.
- H^3PC: Parallelism and representation.

Area, Time (Performance) and Power Design Tradeoffs

$T^3P=k$



AT or $AT^2=k$

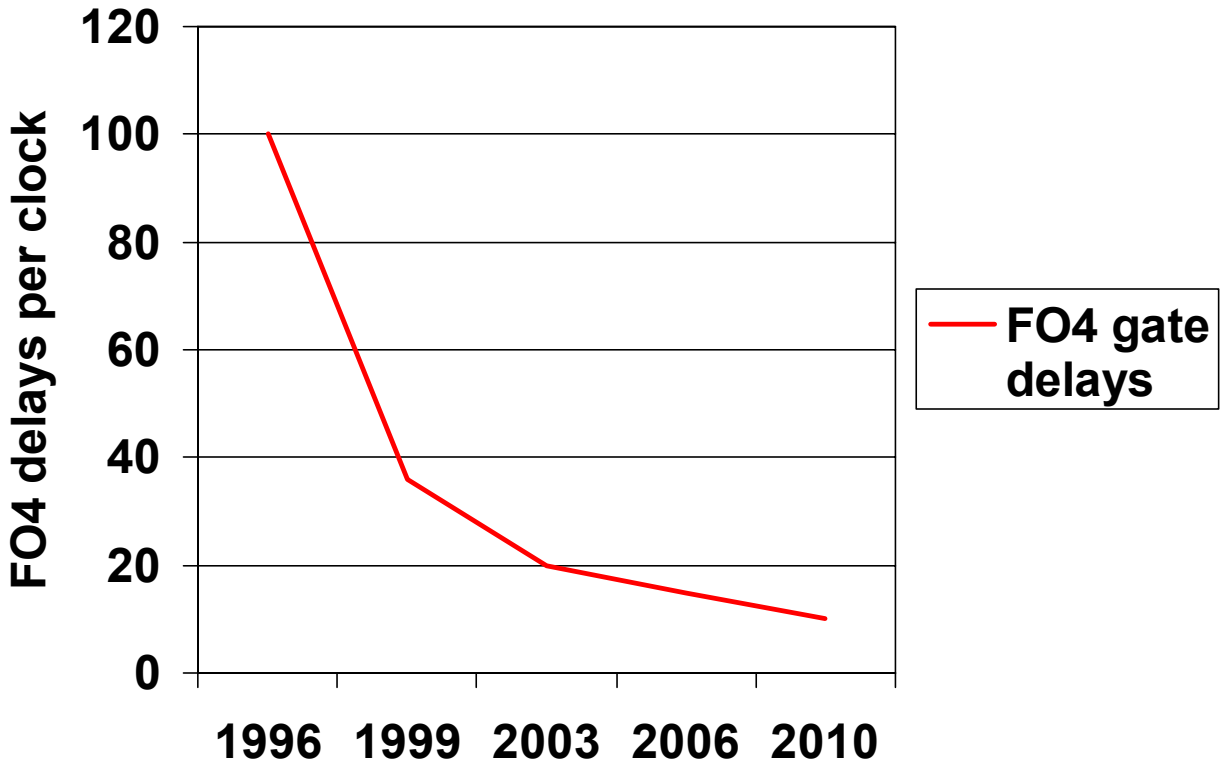
High Speed Clocking

Fast clocks are not based on technology scaling, but on architecture/logic techniques:

  - Smaller pipeline segments, less clock overhead

- Microprocessors increased clock speed more rapidly than (SIA) predicted (from ‘99-04) …fast clocks and short pipe segments)
- But fast clocks do not by themselves increase system performance and they have their own costs.

Changes in pipeline segment size



M.S. Hrishikesh et al., "The Optimal Logic Depth Per Pipeline Stage is 6 to 8 FO4 Inverter Delays." 29th ISCA: 2002.

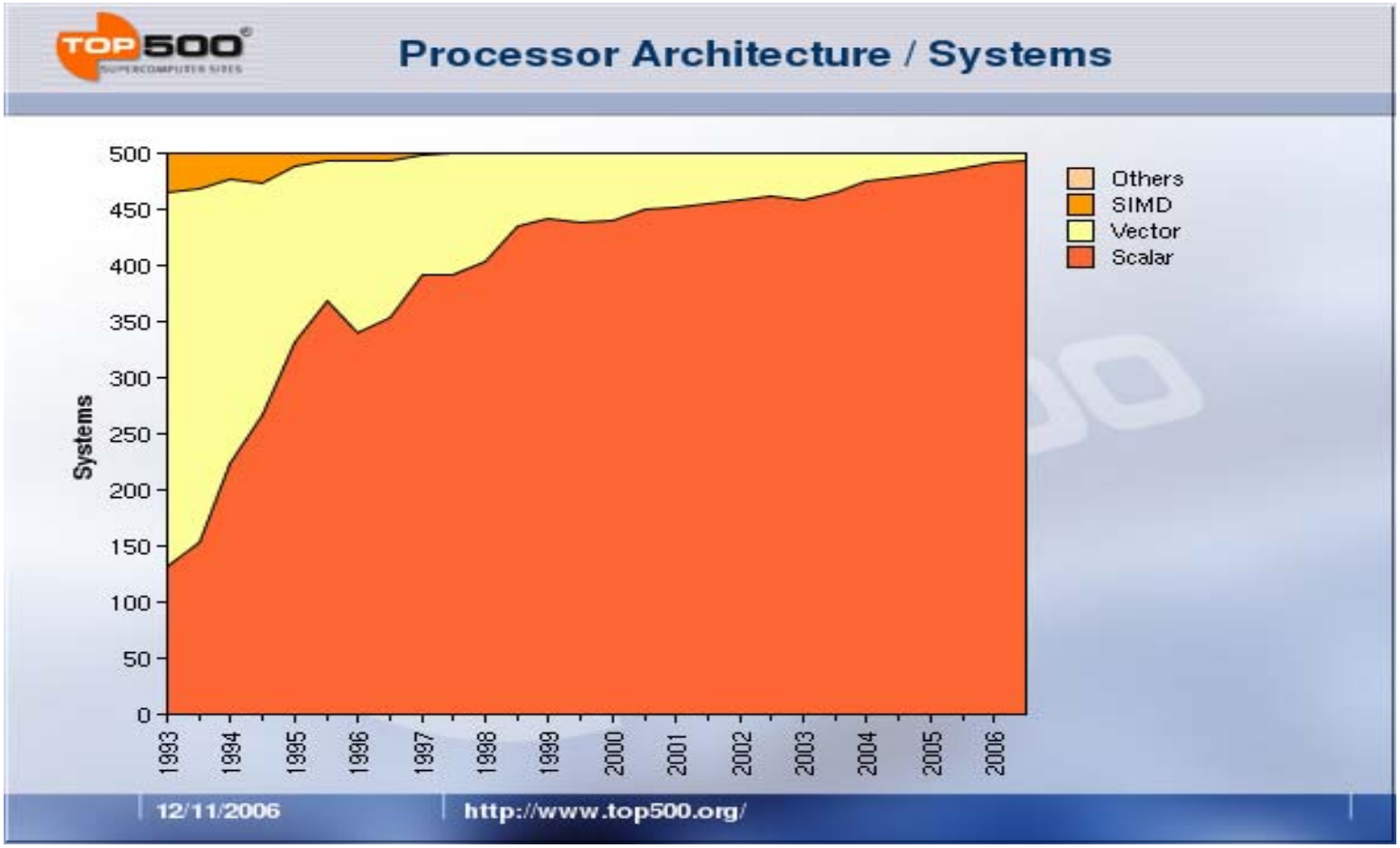
HiPC: Translating time into performance & scaling the walls

- ***The memory "wall"***
- ***The power "wall".***
- ***The segment size limit***
- The only reasonable way forward is multiple concurrent computing elements
- But we got here because of limits on the sequential technology ***NOT*** success with parallel technology

Outline II

- HPC: What's new at the parallel processor node
- H^2PC: And beyond the cluster / node for really big applications;
- H^3PC: Parallelism and representation.

HPC: Trends at the node



The HPC node with symmetric processors

current direction

Multiple processors (each multithreaded) on single die, together with portion of memory space

problems

Large configurations limited by power.

Programming parallel processors to realize corresponding speedup

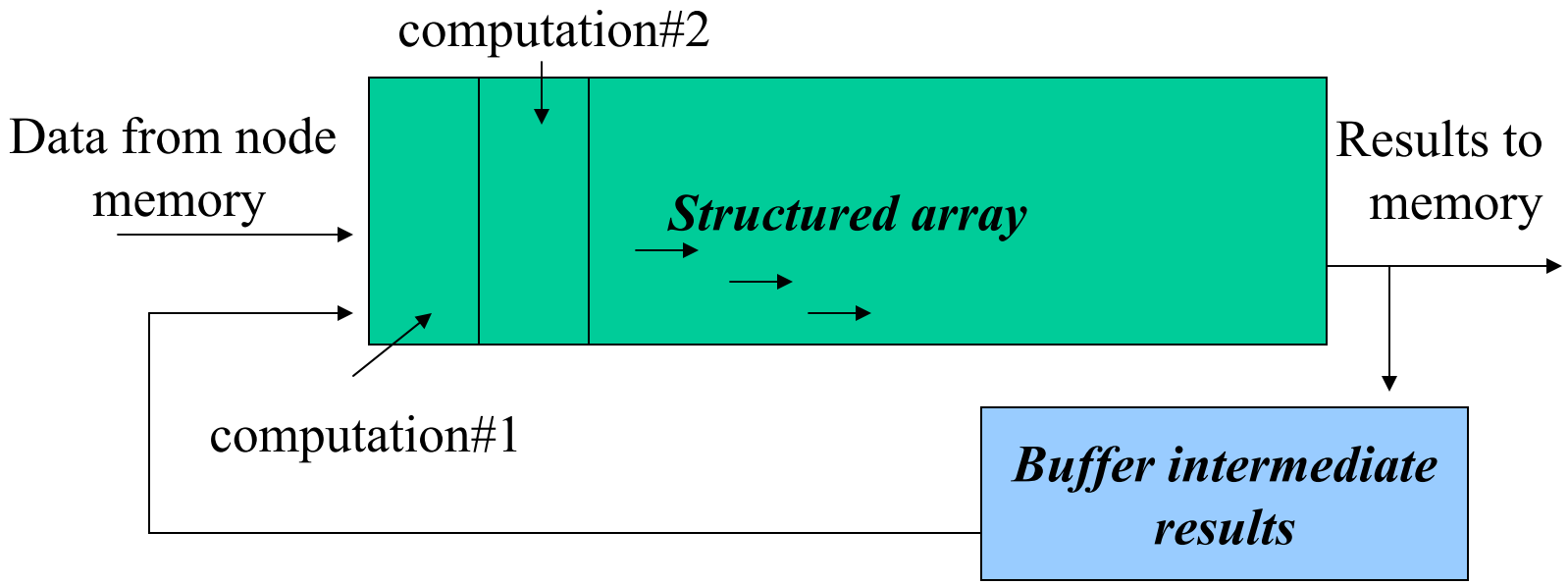
Can quiescent processor / tasks be energy efficient?

HPC: the case for accelerators (heterogeneous processors)

- Traditional HPC processors are designed to optimize task latency, but not large task throughput.
- Multi core/ multi thread offers more limited possibilities.
- Structured arrays (FPGAs, GPUs, hyper "core" or "cell" dies) offer complementary throughput acceleration.
- Properly configured, an array can provide 10x + improvement in arithmetic (SIMD) or memory bandwidth (by streaming or MISD).
- Node memory has better access and bandwidth than inter node memory.

Accelerate large memory intensive tasks

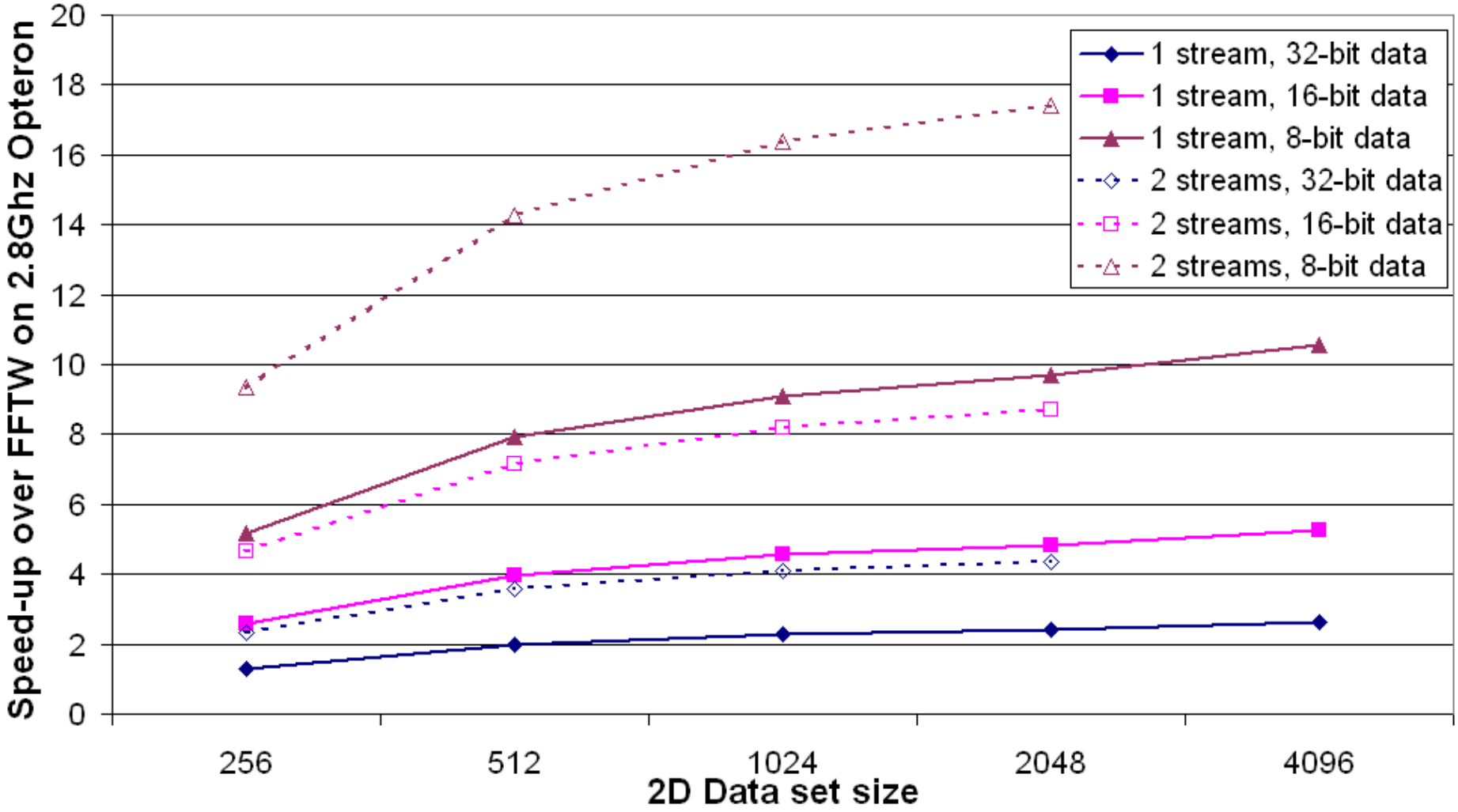
- MISD structured computation: streaming computations across a long array before storing results in memory. Can achieve 100x in improved use of memory.



e.g. FPGA acceleration (Maxeler)

- One tenth the frequency with 10^5 cells per die. The magnitude of parallelism overcomes frequency limitations.
- Advantage is gained by streaming data across large cell array, minimizing memory BW.
- Customized data structures; e.g.17 bit FP; always enough (and not more) precision.
- A software/tools only technology
- Need an in-depth application study to realize acceleration; acceleration is *not* automatic
- Con: (FPGA): fine grain; requires more programming effort.

e.g. Speedup using MAX 1 (FPGA) acceleration



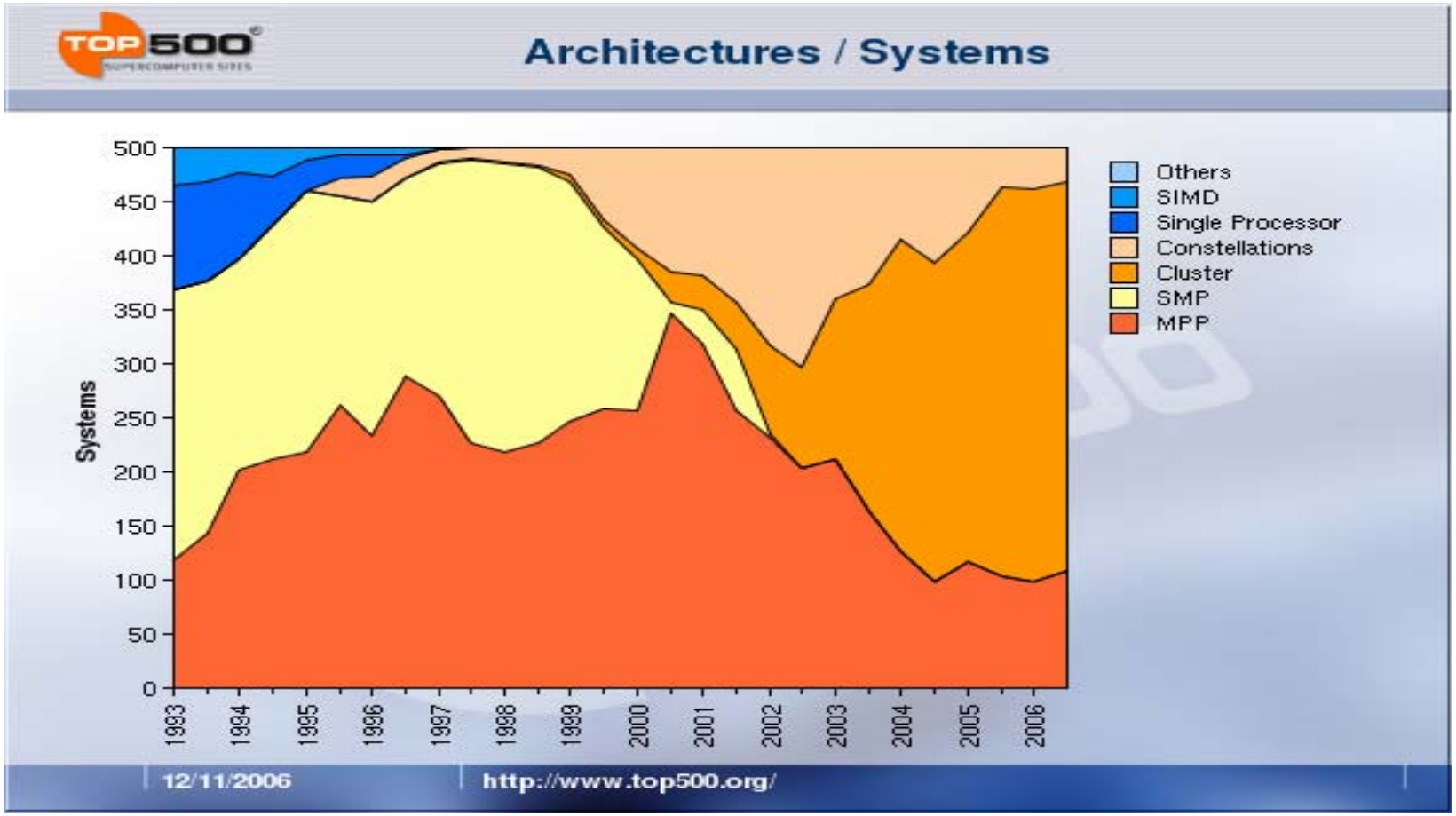
Summary: looking ahead at the node

- Acceleration based speedup using structured arrays with reconfigurable interconnect.
- More attention to memory and/or arithmetic: data compression, streaming, and RAM.
- Lower power with non aggressive frequency use.
- Programming still resembles sequential model; but speedup requires lots of low level program optimization. Good tools are in short supply

Outline III

- H^2PC: And beyond the cluster / node for really big applications.
- H^3PC: Parallelism and representation..

H²PC: Trends in parallel architecture



Some H²PC parallel processors of 40-50 years ago

pp system	*nodes*	*type*
NBS Pilot (Leiner et al, IFIPS '59)	4	Message passing
various Multics, IBM "test and set" instruction, '63	2 (typical)	Shared memory
UNIVAC (precursor to CLC) (Lewis & Mellen, Symp Microelectonics. and Large Sys., '64)	25	Message passing
CLC (BSTJ '70)	10	both

Lessons of the CLC

- 10 processors with 16x2 memory modules
- First use of “software pipelining”.
- Application: (critical) real time “transaction” management and control
- Parallel programming is hard if you want the performance: 4,000 man years of effort! For 2 million lines of code.
- Bigger multi processors are less reliable than smaller configurations!

Other lessons

- Good ideas, by themselves, don't give speedup:
  - Functional programming
  - Improved synchronization or consistency models
  - Shared memory MP and MPP
  - Transactional memory

Other lessons

- Speculation: programming models and practice found efficient for serial processors are probably not efficient for parallel processors.
  - Layers of abstraction hide critical sources of and limits to efficient parallel execution
  - Speedup is achieved by understanding the whole process; application down to the gates

Parallel programming models

- The human mind, traditional math formulations, the programming model and the sequential processor are all *sequential*
- For parallel processors unless the application is already parallel we must transform/translate into a parallel form
- BUT which kind (SIMD, etc.) and with what communications and schedule?

Parallel programming models

- In the absence of workable parallel abstractions the alternative is generalizations based on individually optimized applications and application classes.
- This is a vertical type of generalization rather than the horizontal layered approach of sequential programming.
- Rings of cylinders instead of layered strata.

Parallel programming issues for speedup

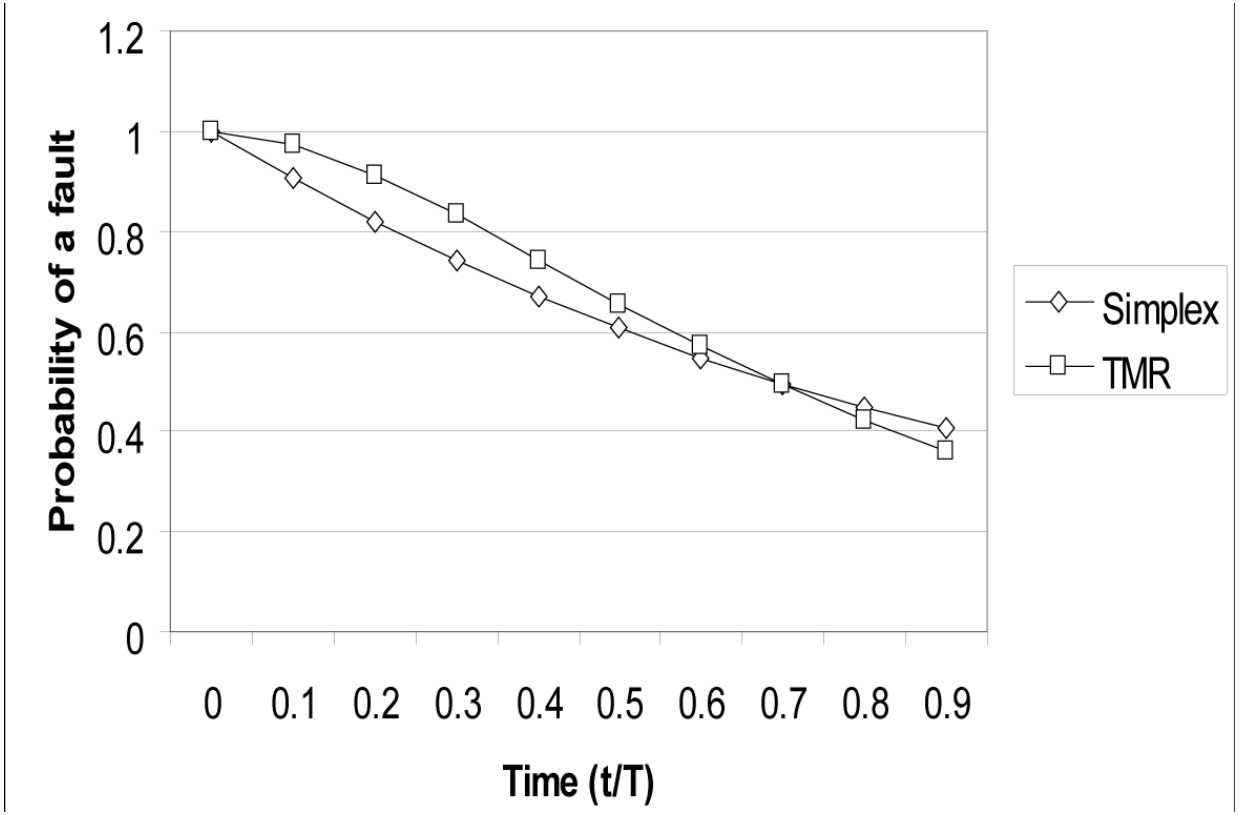
- Type of parallelism
- Control structure and schedule
- Memory model and access time
- Interconnect model and delay
- Arithmetic intensive applications
- Power in large configurations
- Combination

Multiprocessor reliability

- At small feature sizes (e.g. 32 nm) large fields promote electro migration and dielectric fatigue.
- In performance oriented system if any processor fails and the system fails.
- More hardware, more failures
- Use of massive amounts of commodity hardware is a major reliability problem

More hardware, less reliability

- Even TMR has its limitations; as time gets close to T, the expected processor MTBF, simplex is more reliable.



More reliable systems

- Validation of HW/SW
- Error self detect/ self correct
- Transparent, efficient reconfiguration
- Security

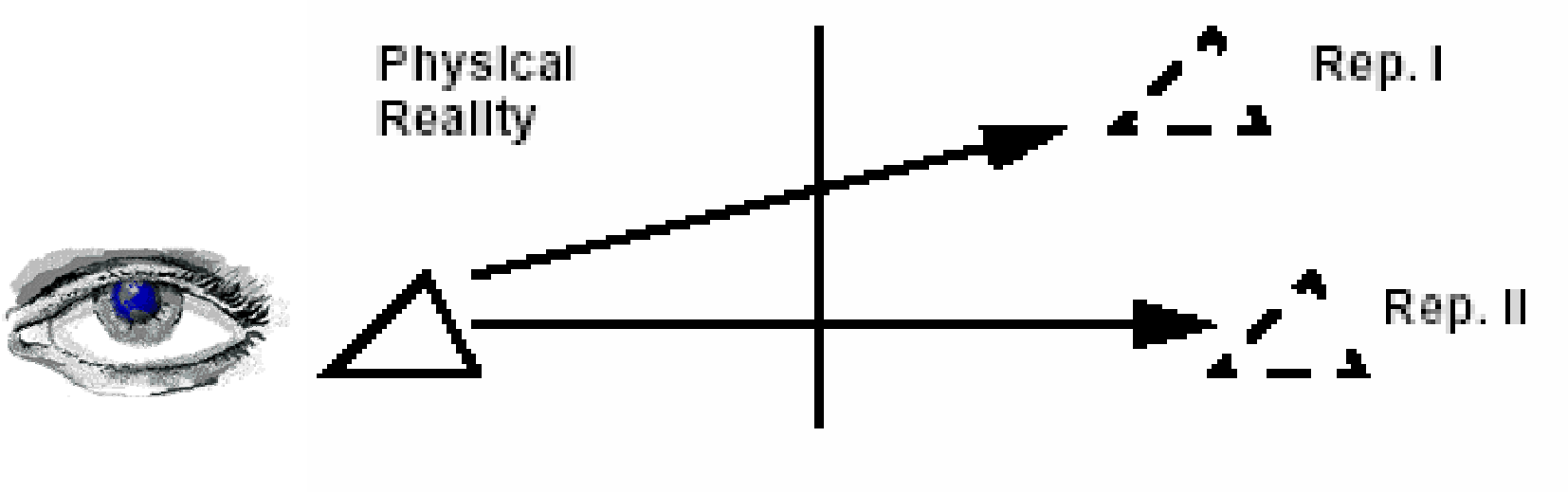
Outline IV

- H^3PC: Parallelism and representation.

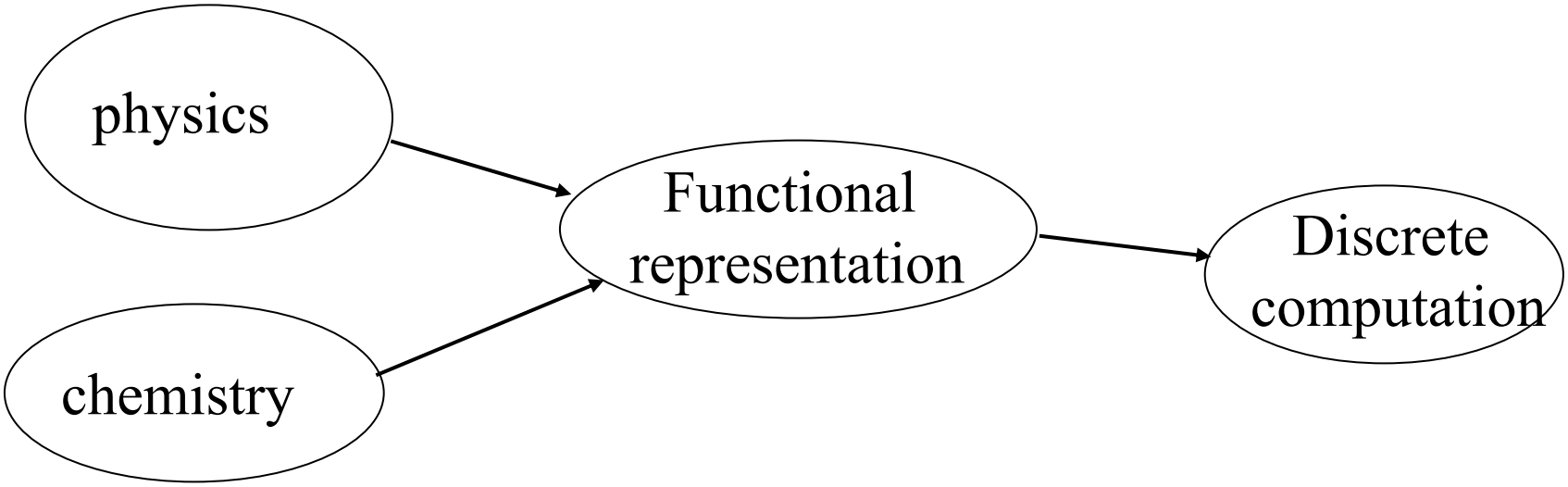
The speedup mismatch in as seen from the application

- What we'd like: messages that have
  - Local communications only
  - Structured communications
  - Short data packets
- What we have: global data memory
  - Large
  - Shared by all nodes
  - Consistent data image

Choosing a representation



Typical scenario for computing results



concept (physical behavior)

continuous function

numerical algorithm

A few models for computation

- *Shared global memory*: data distributed across all nodes
- *"Systolic" computation*: results stream directionally across a grid of nodes.
- "*Cellular" computation*: results communicated only within a node's neighborhood.

e.g. Representation in fluid flow (CFD)

A differential equation (for example) assumes a shared memory model. Fluid flow computation uses the memory limited Navier- Stokes

The particle simulation model matches the data and control flow of the "cellular" model (local interconnects only). The cellular model avoids the memory bottleneck. Such a model was developed by Henon et al in the '90s for restricted CFD.

A cellular implementation of CFD

- Alge was a FPGA based implementation of the Henon cellular CFD
- All cellular machines are not alike; simple cells with 6 nearest neighbors were computationally useless.
- Computational work increases as R^4; R is the Reynolds number
- Interesting CFD involves large R and/or trans sonic motion.

ALGE implementation

- A robust implementation used 26 neighbors with 24 state bits
- ALGE uses particle and momentum conservation and isometric symmetries to simplify cell structure
- Single cell corresponds to about 8k gates and was realized in an FPGA.
- Cells organized as 3D Torus. Performance is 1/frequency rate.

So, how to move to a new representation?

- The mind doesn't reason in a streaming or in a cellular fashion.
- The basic laws of physics and chemistry apply both globally and locally
- Can we transform augmented global functional representations to get more efficient solutions.
- When the same reality is captured by two distinct representations, we need to translate between them.
- Translations maybe possible; e.g. streaming compilers (ASC)

but it may not be easy… Conclusions

- We're moving to multi core, multi processors not because of success with parallel programming but because of failure in scaling sequential processors.
- At the node heterogeneous processors (core plus structured array); tuned to the application seems a good bet.
- Need to create and use a larger variety of node models suited to a range of applications with good Area Time Power efficiency
- The layered sequential programming model is probably not a useful starting point for large scale parallel programming; a better bet is the cylinder model.

but it may not be easy… Conclusions

- There's a lot of research ahead to effectively create parallel translation technology.
- There is the really interesting problem of creating new computational representations… suited to parallel processing and then learning how to interface to them.
- Indeed, just as the field of numerical analysis came into it's own 60 years ago; we now may need to define a new field of representational analysis for parallel processing modeling and translation.