Processing in Memory (II): FlexRAM Intelligent Memory

Instructor: Josep Torrellas

CS533

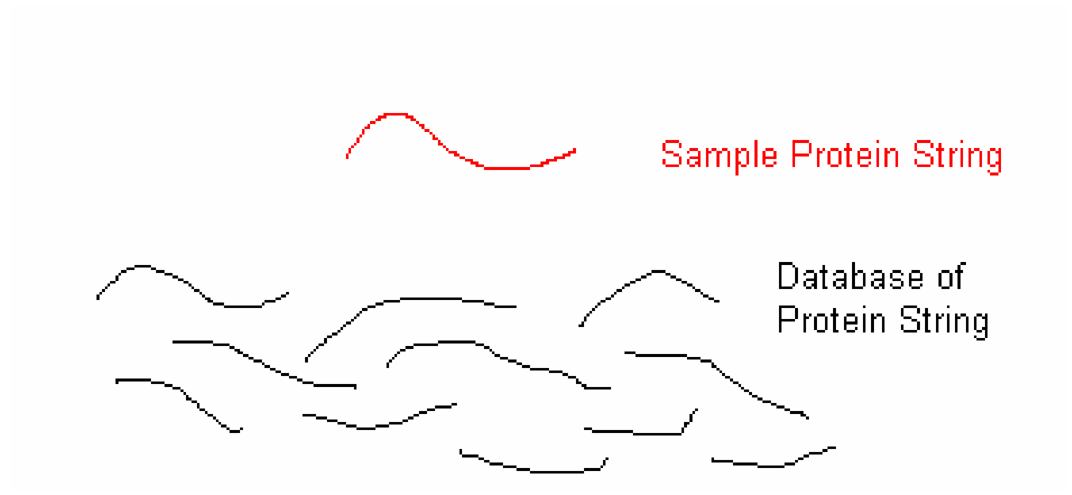
Rationale

- Large & increasing speed gap ⇒bottleneck for many apps.
- Latency hiding bandwidth regaining techniques: diminishing returns
  - out of order
  - lockup free
  - large cache, deep hierarchies
- P/M integration: latency, bandwidth

Key Applications

- Data Mining (decision trees and neural networks)
- Computational Biology (protein sequence matching)
- Multimedia (MPEG-2 Encoder)
- Decision Support Systems (TPC-D)
- Speech Recognition
- Financial Modeling (stock options, derivatives)
- Molecular Dynamics (short-range forces)

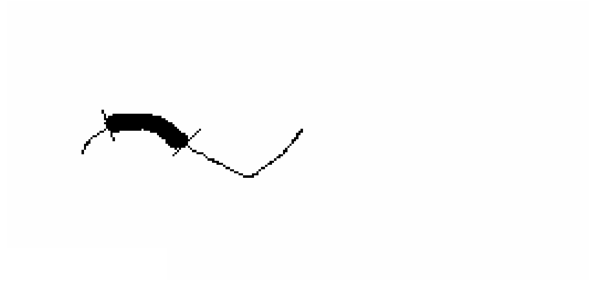
Example App: Protein Matching



- Problem: Find areas of database protein chains that match (modulo some mutations) the sample protein chains

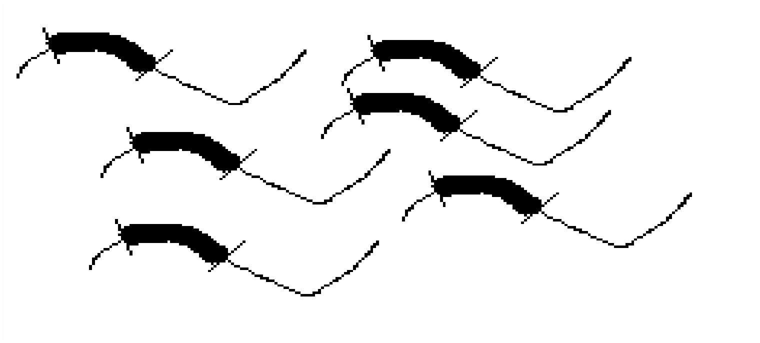
How the Algorithm Works

⌘Pick 4 consecutive amino acids from sample



GDSL

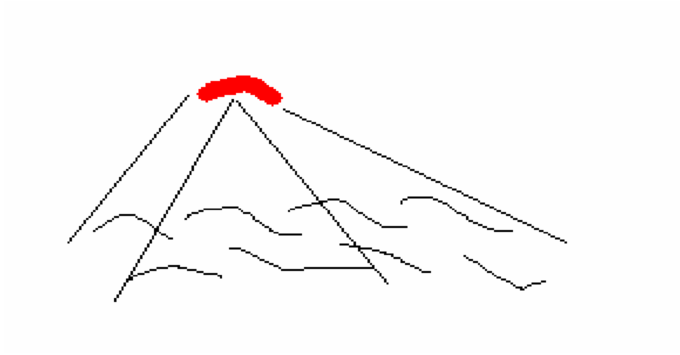
⌘Generate most-likely mutations



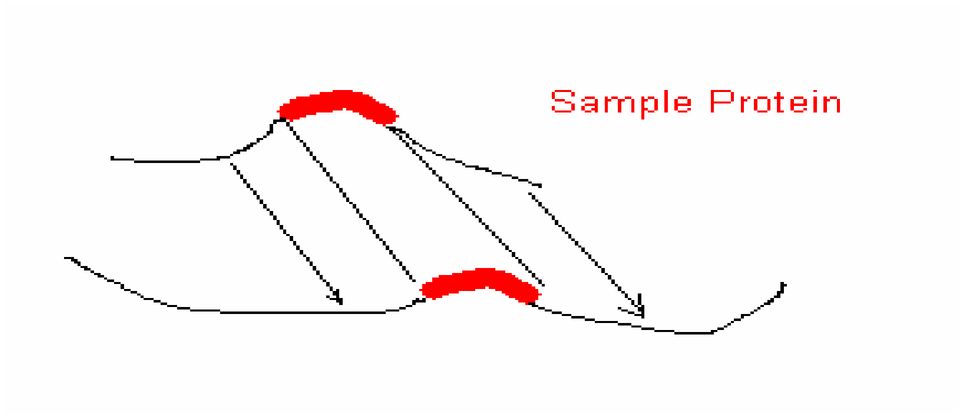
GDSI	GDSM
ADSI	AESI
AETI	GETM

Example App: Protein Matching

⌘ Compare them to every positions in the database proteins



⌘ If match is found: try to extend it



How to Use MLD

- Main compute engine of the machine
  - Add a traditional processor to DRAM chip ⇒ Incremental gains
  - Include a special (vector/multi) processor ⇒Hard to program

UC Berkeley: IRAM

Notre Dame: Execube, Petaflops

MIT: Raw

Stanford: Smart Memories

How to Use MLD (II)

- Co-processor, special-purpose processor
  - ATM switch controller
  - Process data beside the disk
  - Graphics accelerator

Stanford: Imagine

UC Berkeley: ISTORE

How to Use MLD (III)

- Our approach: replace memory chips
  - PIM chip processes the memory-intensive parts of the program

Illinois: FlexRAM

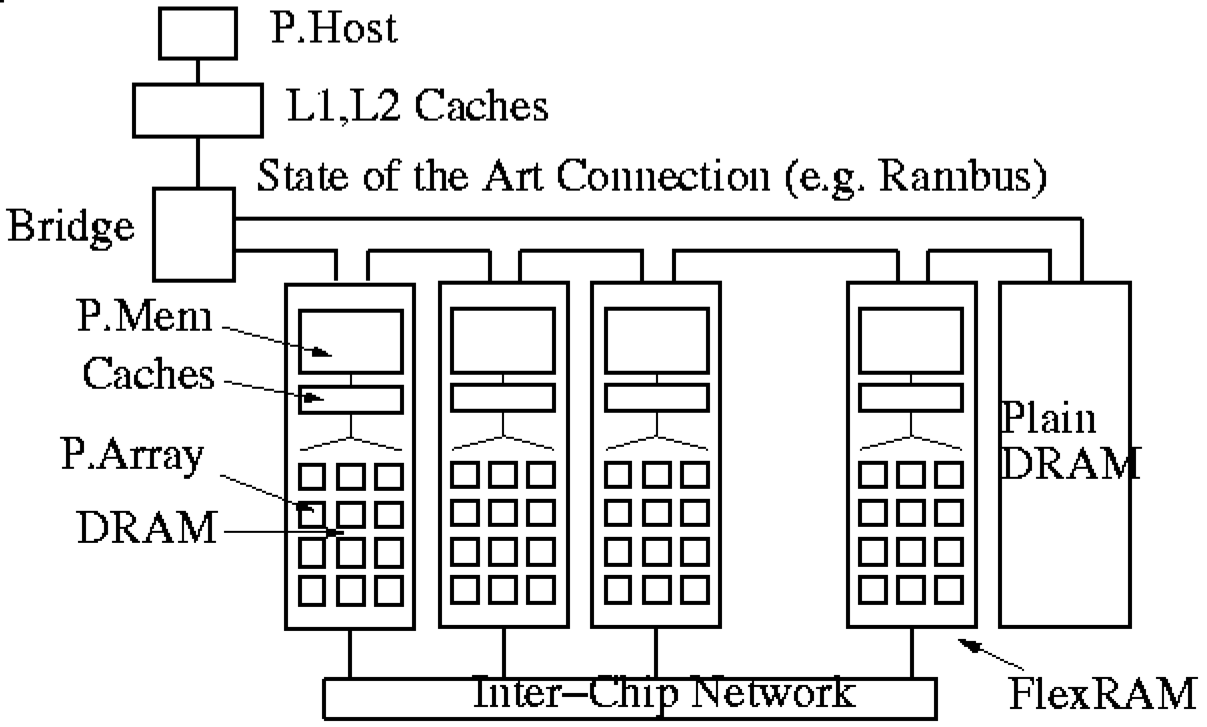
UC Davis: Active Pages

USC-ISI: DIVA

Our Solution: Principles

- Extract high bandwidth from DRAM:
  - Many simple processing units
- Run legacy codes with high performance:
  - Do not replace off-the-shelf μP in workstation
  - Take place of memory chip. Same interface as DRAM
  - Intelligent memory defaults to plain DRAM
- Small increase in cost over DRAM:
  - Simple processing units, still dense
- General purpose:
  - Do not hardwire any algorithm. No Special purpose

The FlexRAM Architecture

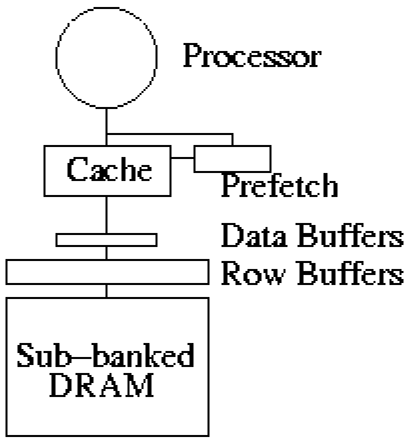


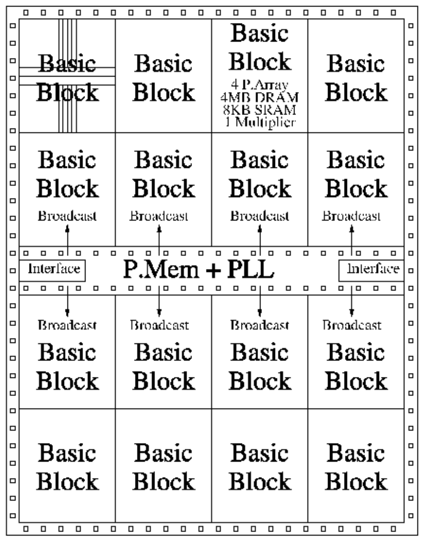
Yi Kang, Wei Huang, Seung-Moon Yoo, Diana Keen, Zhenzhou Ge, Vinh Lam, Pratap Pattanaik and Josep Torrellas - ICCD99

MORE UP TO DATE: “FlexRAM Architecture Design Parameters”, S.Yoo, J.Renau, M.Huang, and J. Torrellas. Tech Report 1584, Oct 2000. http://iacoma.cs.uiuc.edu/papers.html

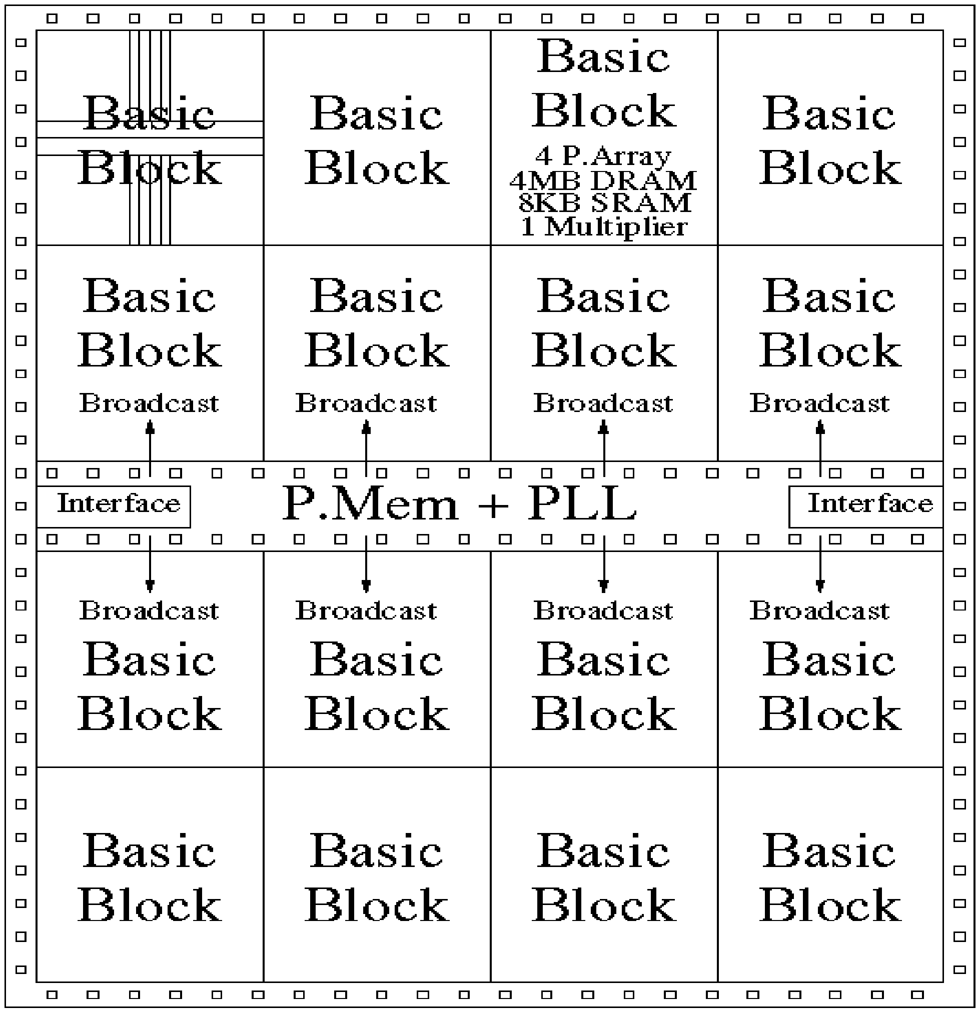
Chip Architecture

- 64 nodes, each one includes:
  - 2-issue processor @800Mhz
  - 1MByte DRAM (12 clk)
  - Row Buffers (6 clk)
  - Cache (1 clk)

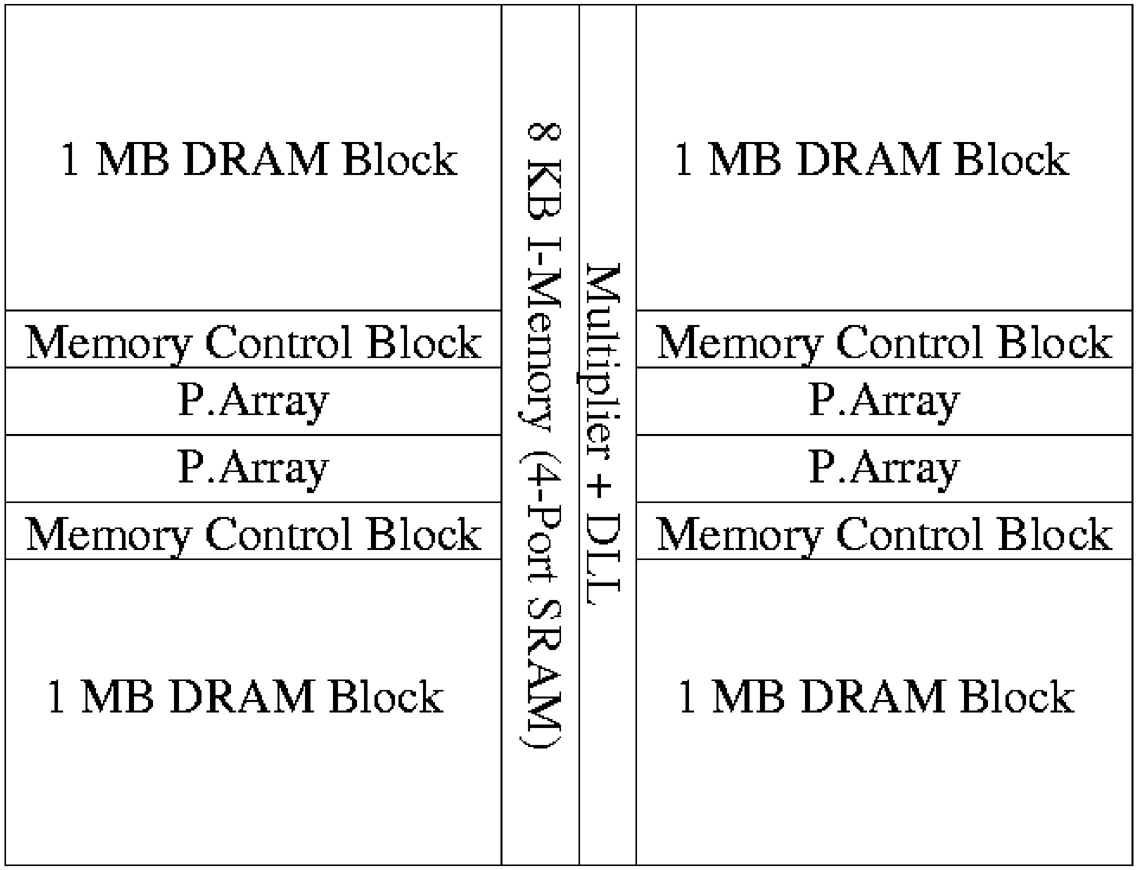




Chip Layout


Basic Block
Basic Block
Basic Block
4 P.Array
4MB DRAM
8KB SRAM
1 Multiplier
Basic Block
Basic Block
Basic Block
Basic Block
Basic Block
Broadcast
Broadcast
Broadcast
Broadcast
Interface
P.Mem + PLL
Interface
Broadcast
Broadcast
Broadcast
Broadcast
Basic Block
Basic Block
Basic Block
Basic Block
Basic Block
Basic Block
Basic Block
Basic Block

Basic Block


1 MB DRAM Block
Memory Control Block
P.Array
P.Array
Memory Control Block
1 MB DRAM Block
8 KB I-Memory (4-Port SRAM)
Multiplier + DLL
1 MB DRAM Block
Memory Control Block
P.Array
P.Array
Memory Control Block
1 MB DRAM Block

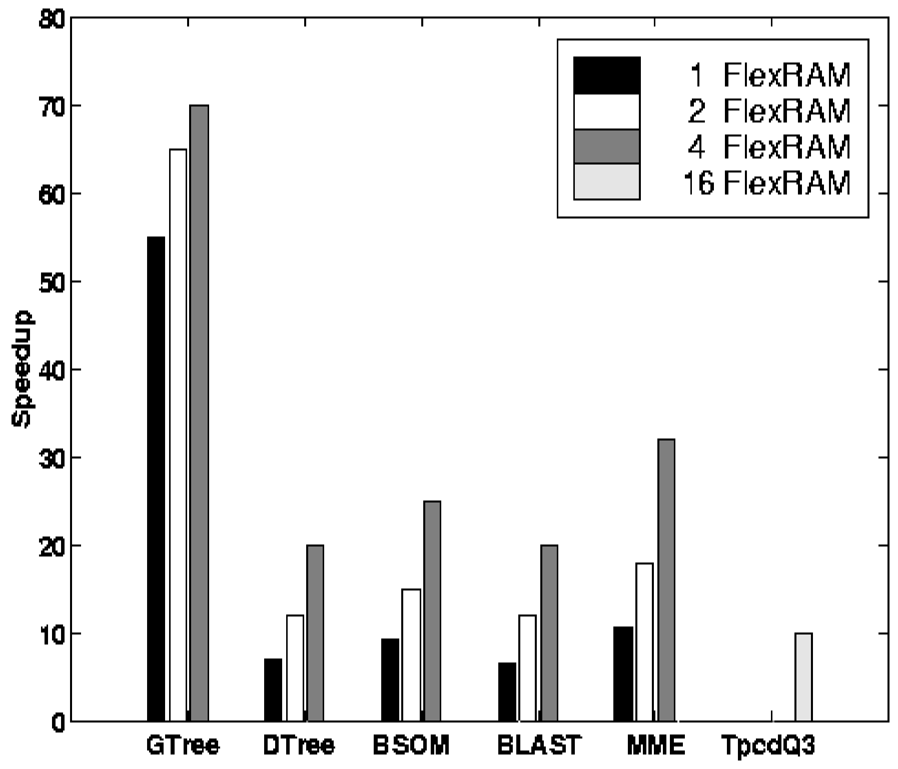
Issues

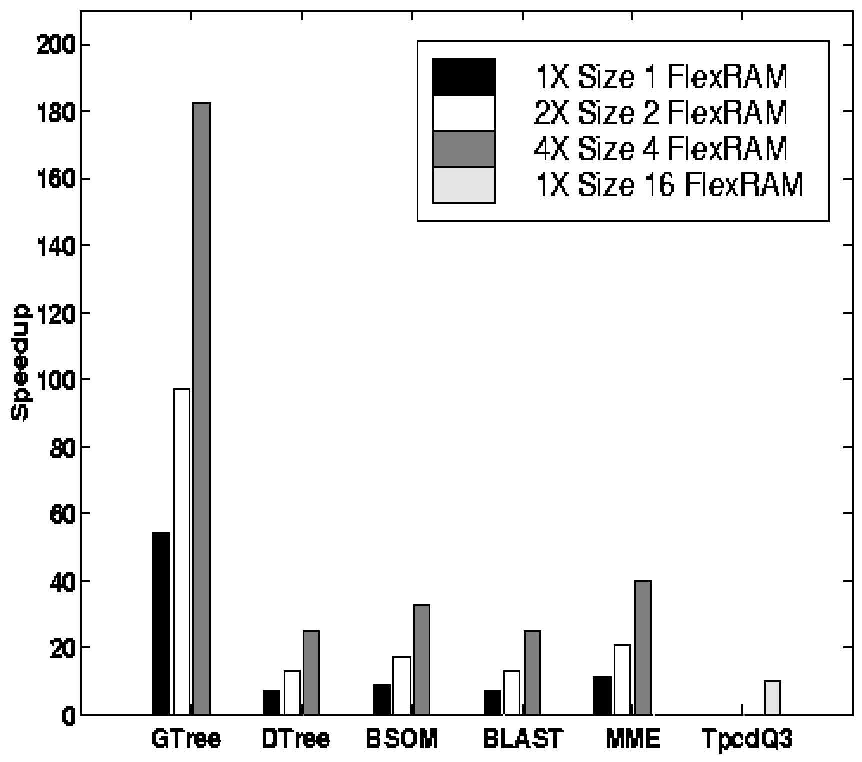
- **Communication P.Mem-P.Host:**
  - P.Mem cannot be the master of bus
  - Protocol intensive interface: Rambus
- **Virtual memory:**
  - P.Mems and P.Arrays use virtual addresses
  - Small TLB for P.Arrays
  - Special page mapping

Evaluation

P.Host	P.Host L1 & L2	Bus & Memory
Freq: 800 MHz	L1 Size: 32 KB	Bus: Split Trans
Issue Width: 6	L1 RT: 2.5 ns	Bus Width: 16 B
Dyn Issue: Yes	L1 Assoc: 2	Bus Freq: 100 MHz
I-Window Size: 96	L1 Line: 64 B	Mem RT: 262.5 ns
Ld/St Units: 2	L2 Size: 256 KB	
Int Units: 6	L2 RT: 12.5 ns	
FP Units: 4	L2 Assoc: 4	
Pending Ld/St: 8/8	L2 Line: 64 B	
BR Penalty: 4 cyc		
P.Mem	P.Mem L1	P.Array
Freq: 400 MHz	L1 Size: 16 KB	Freq: 400 MHz
Issue Width: 2	L1 RT: 2.5 ns	Issue Width: 1
Dyn Issue: No	L1 Assoc: 2	Dyn Issue: No
Ld/St Units: 2	L1 Line: 32 B	Pending St: 1
Int Units: 2	L2 Cache: No	Row Buffers: 3
FP Units: 2		RB Size: 2 KB
Pending Ld/St: 8/8		RB Hit: 10 ns
BR Penalty: 2 cyc		RB Miss: 20 ns

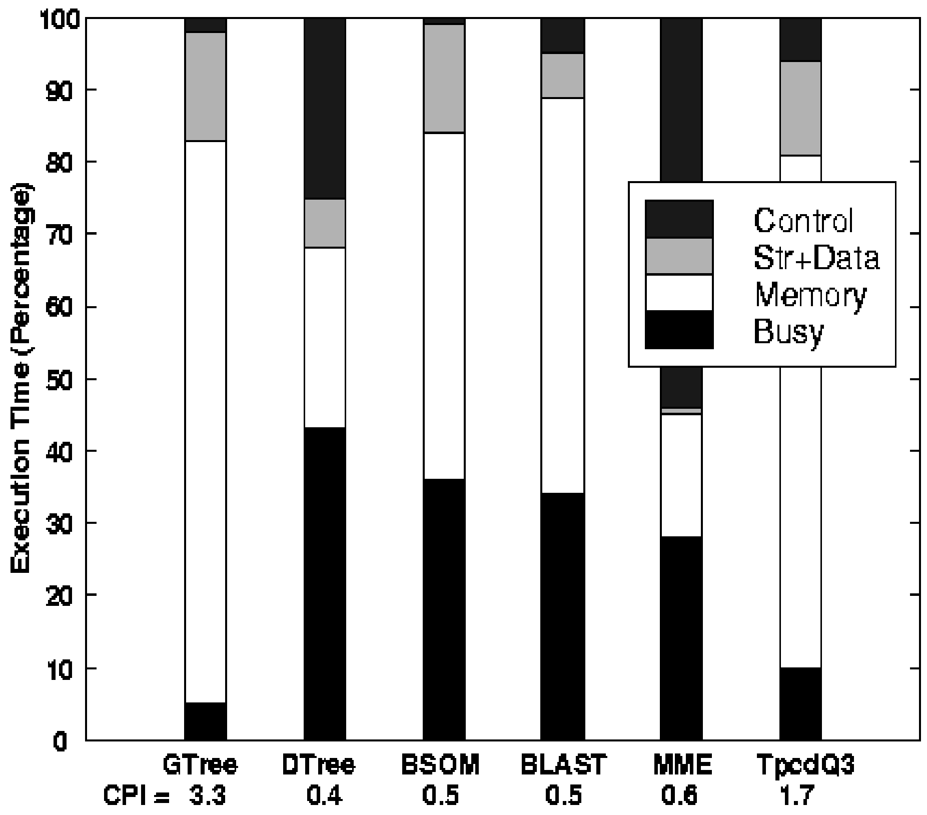
Speedups





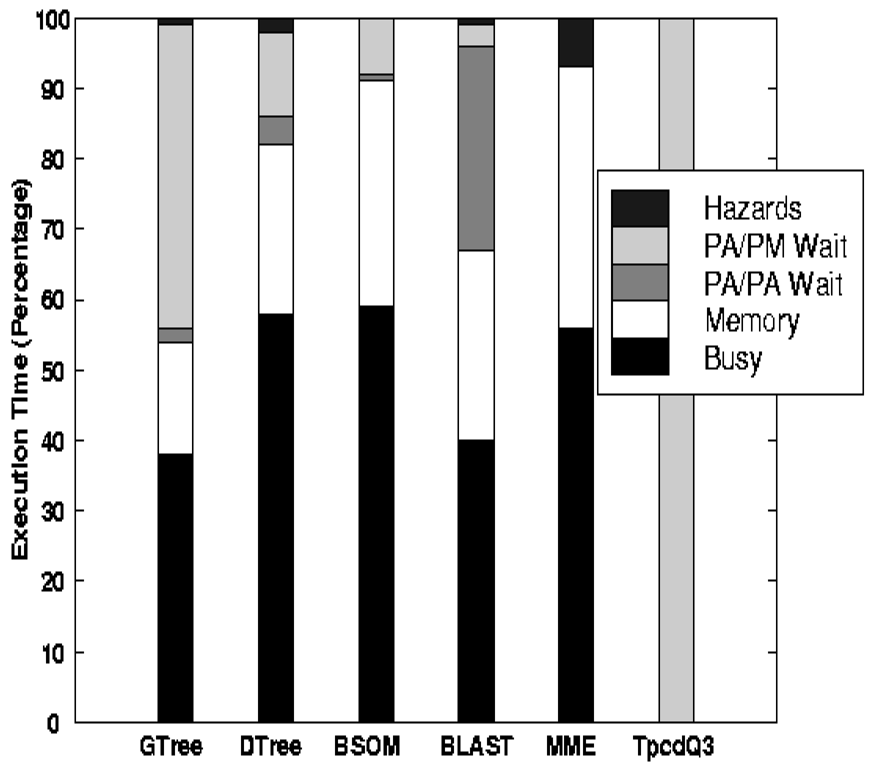
- Constant Problem Size
- Scaled Problem Size

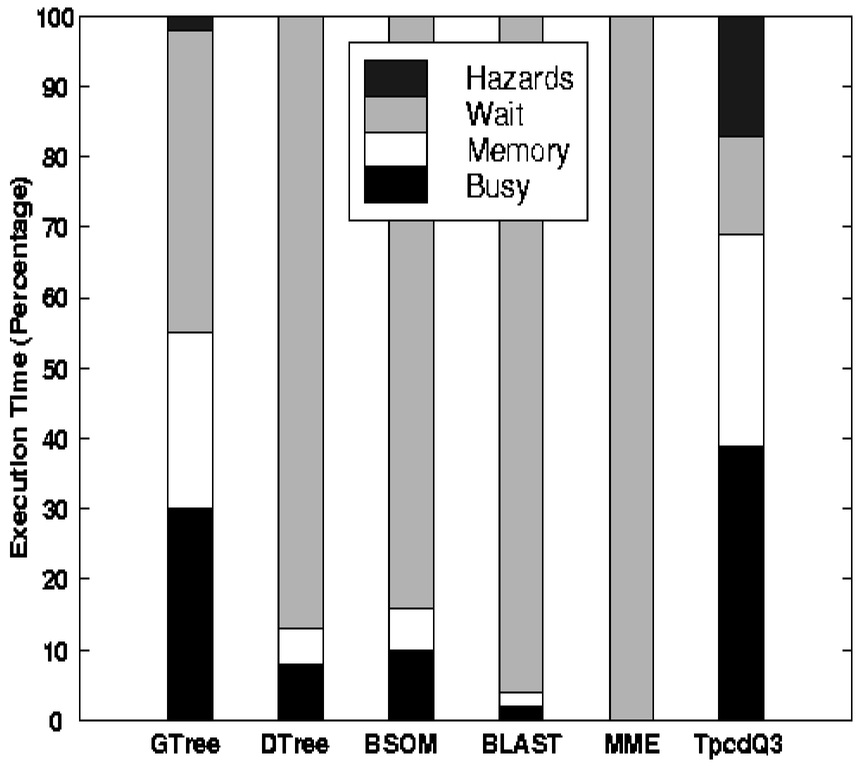
Utilization



⌘ Low P.Host Utilization

Utilization

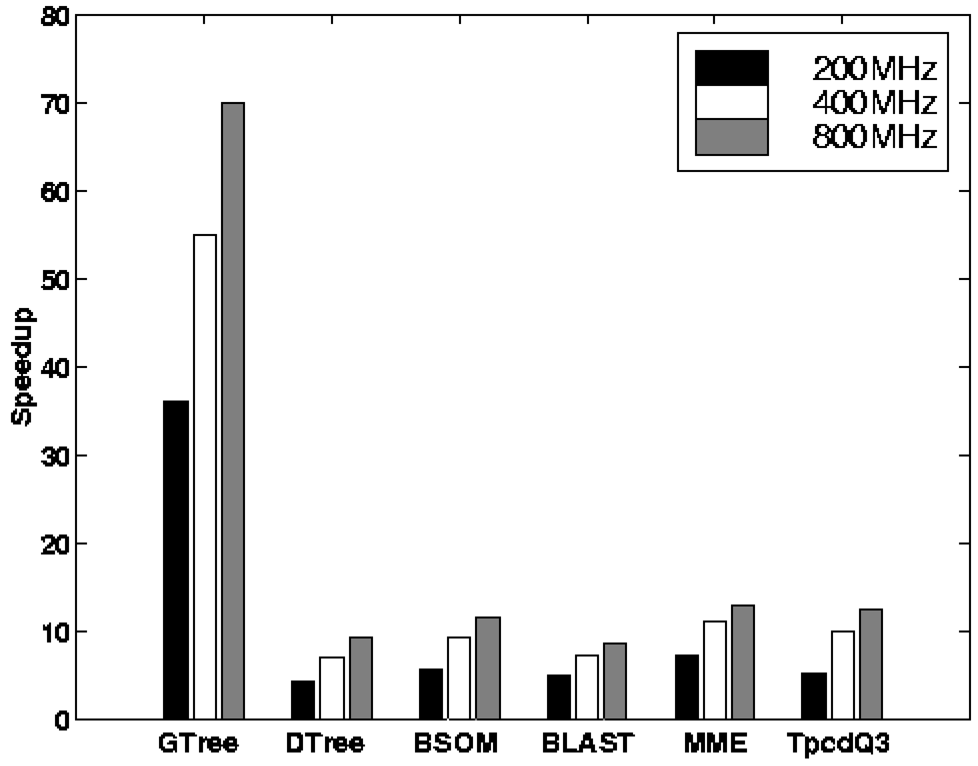




⌘ High P.Array Utilization

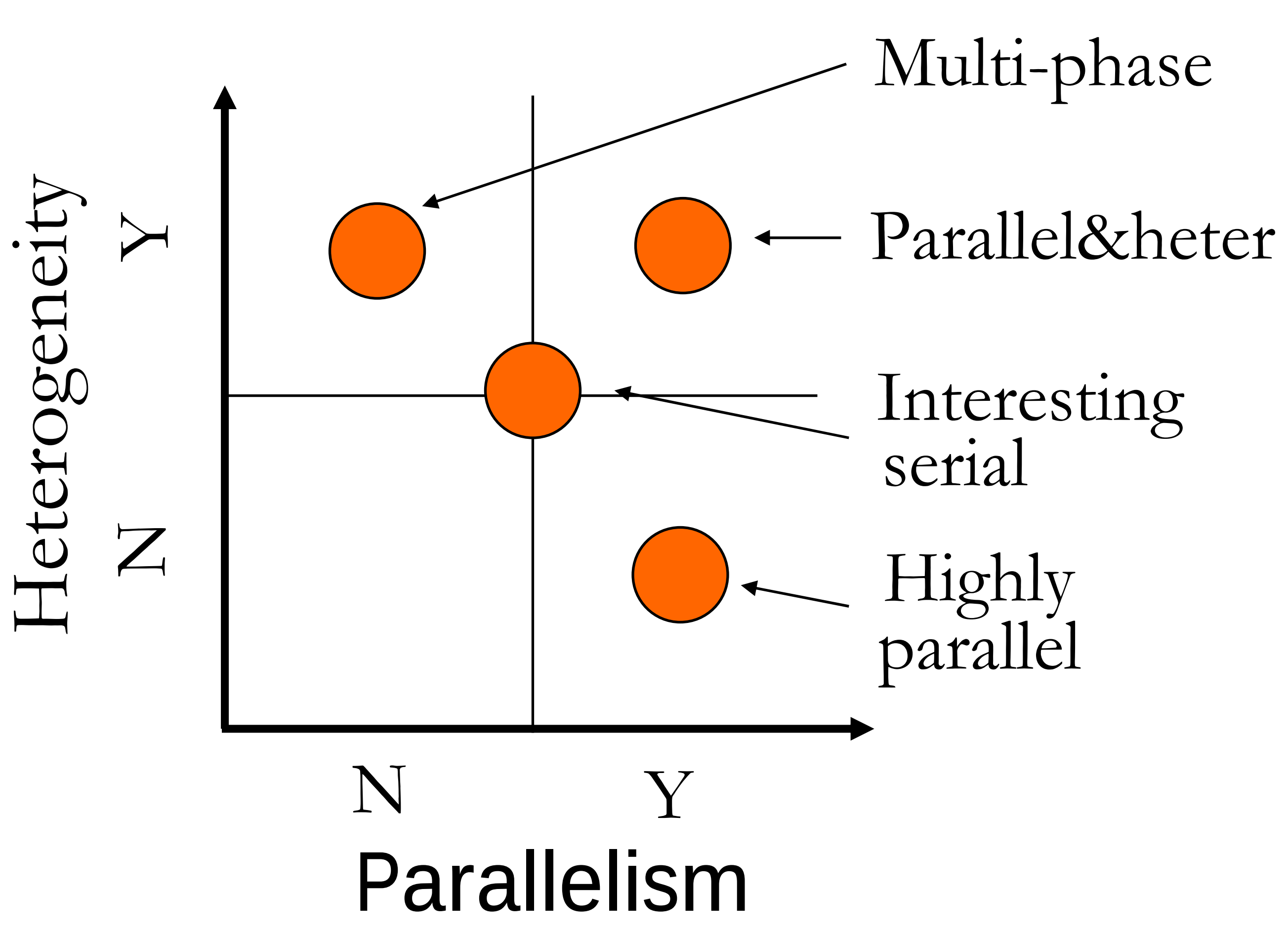
⌘ Low P.Mem Utilization

Speedups



- Varying Logic Frequency

What's It Useful For?



File compress

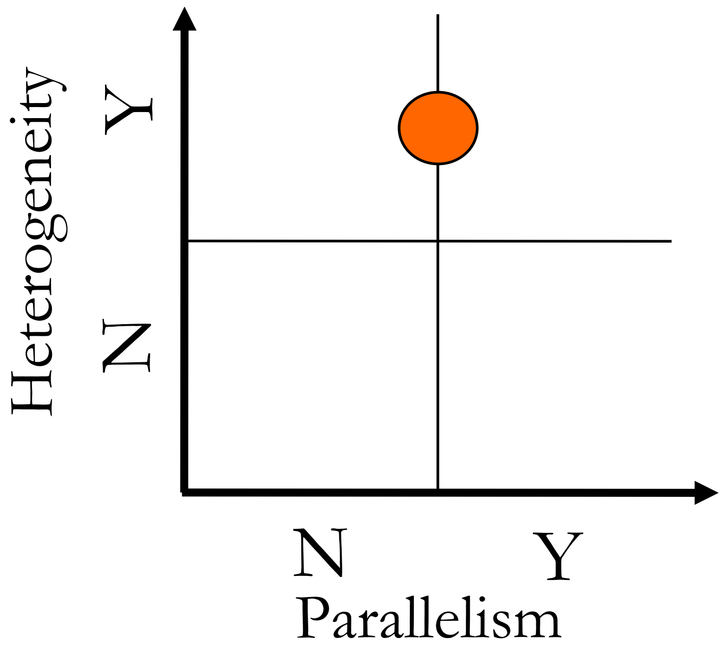
Dbase search

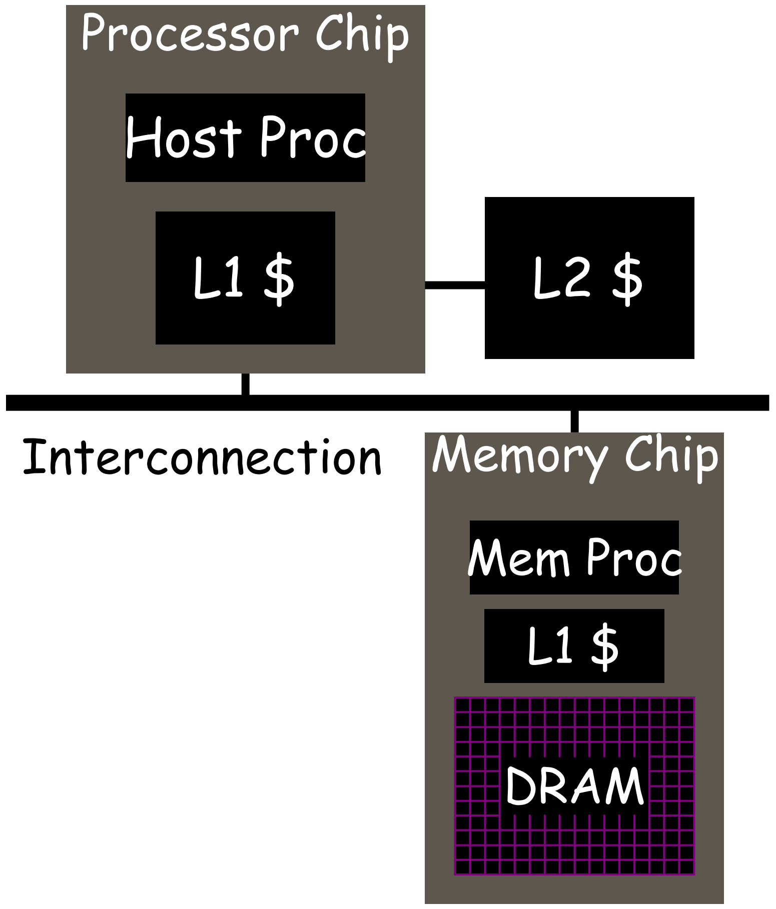
Some SpecInt

Numerical

Example

Automatic Programming





Automatic Algorithm
[HPCA’01]

- Partition code into homogeneous *modules*:
  - Basic partitioning: module = loop nest
  - Advanced partitioning: expand modules and fuse them if same estimated *affinity* [1]
- Map
- Overlap

1. Where it runs best (host proc or mem proc)

Partitioning

```
N1 = N*2
DO I=1, N1
  N2 = X * 4
  DO J = 1, N2
    X = …
    A(J,I) = …
  ENDDO
  IF (X .LT. 1.0) THEN
    X = …
  ENDIF
ENDDO
C(N) = …
DO K = 1, N-1
  B(K) = C(K+1)
ENDDO
```

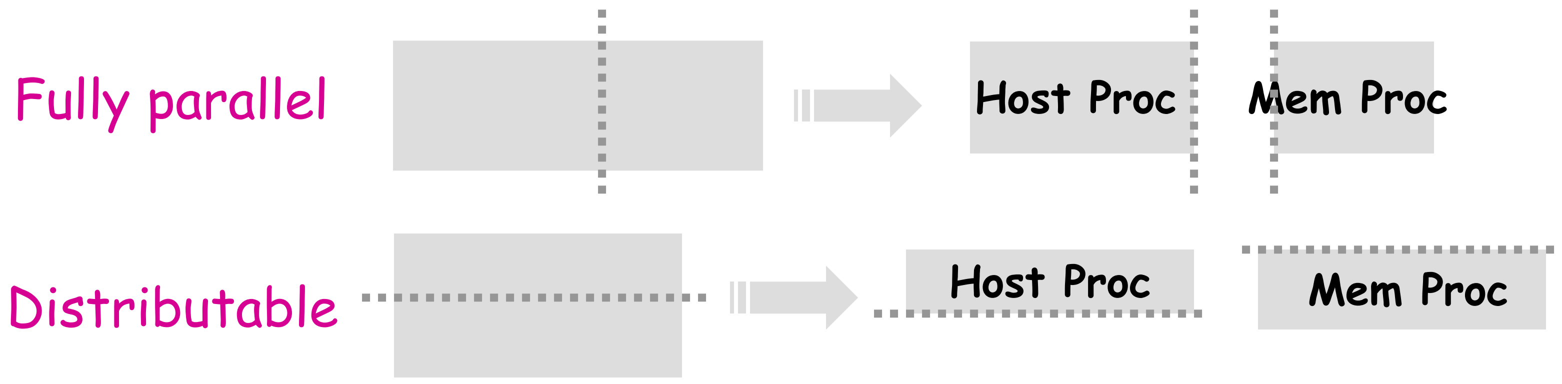
Automatic Algorithm
[HPCA’01]

- Partition
  - Basic partitioning
  - Advanced partitioning
- Map modules to procs based on estimated affinity
  - Static
  - Dynamic
- Overlap

Automatic Algorithm
[HPCA’01]

- Partition
  - Basic partitioning
  - Advanced partitioning
- Map
  - Static
  - Dynamic
- Overlap execution of the two procs

Overlap by Breaking up a Module



Dopipe

```
do i=1,100
   A(i) = A(i-1)+B(i)
   C(i) = A(i)
enddo
```

Host Proc

```
do i=1,100
   A(i) = A(i-1)+B(i)
   if mod(i,4)=0 then
      Writeback
      Signal
   endif
enddo
```

Mem Proc

```
do i=1,100
   if mod(i+3,4)=0 then
      Wait
   endif
   C(i) = A(i)
Enddo
```

Overall Speedups

Host

800MHz, 6-issue ooo
32KB L1, 1MB L2
160 cycle mem

Mem

800MHz, 2-issue io
16KB cache
21 cycle mem

Apps.	Host Alone / Flex No Overlap	Host Alone / Flex Overlap
Swim	1.67	2.71
Tomcatv	1.17	1.60
LU	1.26	1.22
TFFT2	1.42	1.22
Mgrid	1.05	1.55
Average	**1.31**	**1.66**
Bzip2	1.37	-
Mcf	1.37	-
Go	0.97	-
M88ksim	1.01	-
Average	**1.18**	**-**

Overall Speedups

Host

800MHz, 6-issue ooo

32KB L1, 1MB L2

160 cycle mem

Mem

800MHz, 2-issue io

16KB cache

21 cycle mem

Apps.	Host Alone / Flex No Overlap	Host Alone / Flex Overlap	Amdahl's 2 Hosts
Swim	1.67	2.71	2.00
Tomcatv	1.17	1.60	1.67
LU	1.26	1.22	1.04
TFFT2	1.42	1.22	1.91
Mgrid	1.05	1.55	1.94
Average	**1.31**	**1.66**	**1.71**
Bzip2	1.37	-	1.01
Mcf	1.37	-	1.01
Go	0.97	-	1.01
M88ksim	1.01	-	1.03
Average	**1.18**	**-**	**1.02**

⌘ Speedups are comparable & higher than ideal

Intelligent Memory Ops (IMOs)

Array IMOs	Example
Element-by-element (with optional mask): *add, mpy, and, or*	x(i) = (y(i)+z(i)) mask w(i)
Reduction: *sum, product, or, and, minvalue, maxvalue*	x = or(y(i))
Recurrence: *linear first order, linear second order*	x(i)=a(i)*x(i-1)+b(i)
Manipulation: *rotate, transpose, tile*	x(i,j)=transpose(y(i,j))
Set IMOs	**Example**
Set to set: *union, intersection, difference, equal*	S3 = union(S1,S2)
Element to set: *in, from, include,remove,exists,forall*	Bool = in(x,S1)