

Hardware Profile-guided Automatic Page Placement on ccNUMA systems

Jaydeep Marathe, Frank Mueller



North Carolina State University, Dept. of CS

Summary

Key Idea

- Target ccNUMA systems
- Reduce **average** access latency - Allocate pages "closer" to nodes with most frequent access.

Approach

- Whole-program profiling → Approximate memory trace.
- Itanium2 H/W → Filter cache hits → ↓ tracing overhead
- Use Trace to decide *page affinity*.

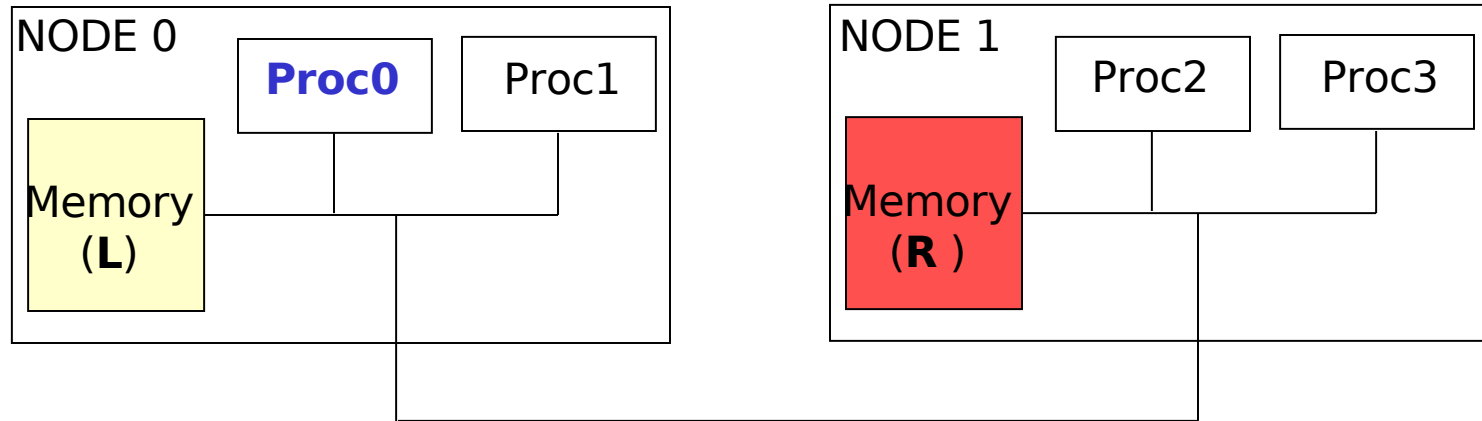
Contributions

- 2 H/W profile sources → long-latency loads, DTLB misses
- Transparently handles *static(bss)* & *dynamically allocated* memory
- > 20% Wallclock improvement for several benchmarks

Automated, low-overhead scheme, good wallclock improvements

The Opportunity

NUMA = *Non-Uniform* Memory Architecture



- Memory access takes longer if memory is remote.
- On *SGI Altix*:

Proc0 → Local Memory L : 207 cycles

Proc0 → Remote Memory R : 409 cycles

Allocating Memory Near Computation → *Big Performance Impact.*

Environment

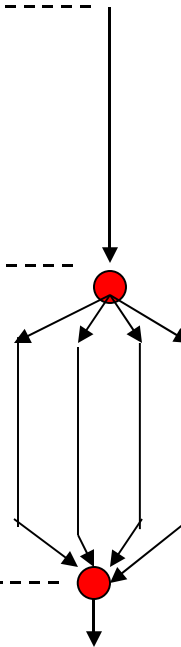
- SGI Altix ccNUMA system, Itanium2 processors.
- Node: 2 Procs + Physical Memory
- OS (Linux 2.4) only supported **First-Touch Page Allocation**.
- **Scientific Computing workloads**, bound OpenMP threads.

```
//1. Init  
for(l=0;l < MATDIM;l++)  
    A[l] = l;
```

```
//2. Work  
#pragma omp parallel for  
for(l=0;l < MATDIM;l++)  
    A[l] = A[l]*B[l];
```

Serial: Only Thread0

Parallel: Many threads



OpenMP's "fork-join" parallelization model

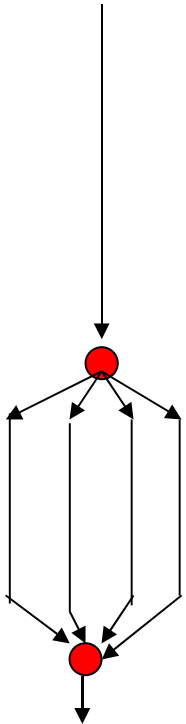
Bad Page Placement: What's wrong here ?

```
//init
for(l=0;l < MATDIM;l++)
  A[l] = l;
```

Only Thread0 (node0) alive, A[] allocated on node0 !

```
//work
#pragma omp parallel for
for(l=0;l < MATDIM;l++)
  A[l] = A[l]*B[l];
```

Remote access (400 cycles) for non-local threads !



- Several examples in [NAS-2.3 C benchmarks](#) (RWCP, Japan).

Large Programs → [Very hard to detect with static compiler analysis](#)

More Subtle Badness

SPEC OMP 2001: [320.quake](#)

//Initialization

```
READ_PACKFILE()
{
  for (i = 0; ....; i++) {
    .....
    fscanf(...., "...", &ARCHmatrixcol[i]);
    while(....)
      ARCHmatrixindex[++oldrow] = i;
  }
}
```

//HOT Function

```
SMVP()
{
  #pragma omp parallel
  {
    ... = ARCHmatrixindex[];
    ... = ARCHmatrixcol[];
  }
}
```

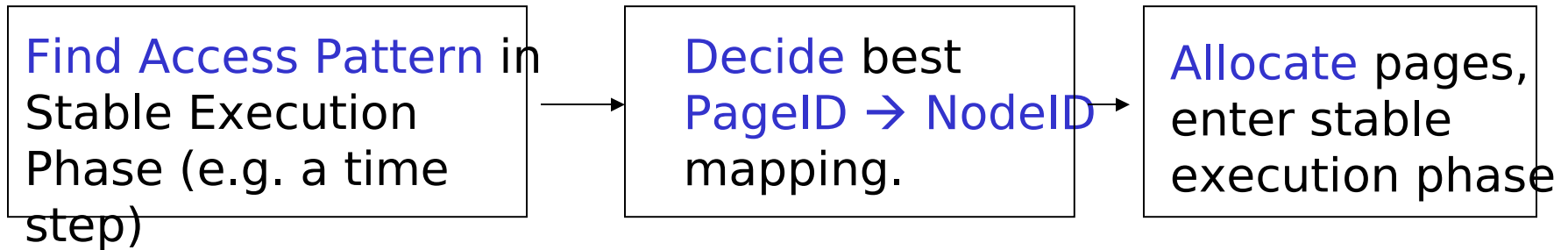
- Memory initialized from file → *Serialized* → ARCHmatrixindex[], ARCHmatrixcol[] allocated on node0.
- 21% increase in wallclock time.

Sometimes initialization is *inherently serial* !

Achieving better page placement

Key: **Mismatch** b/w node where page allocated \leftrightarrow node where page most heavily used.

Ideally:

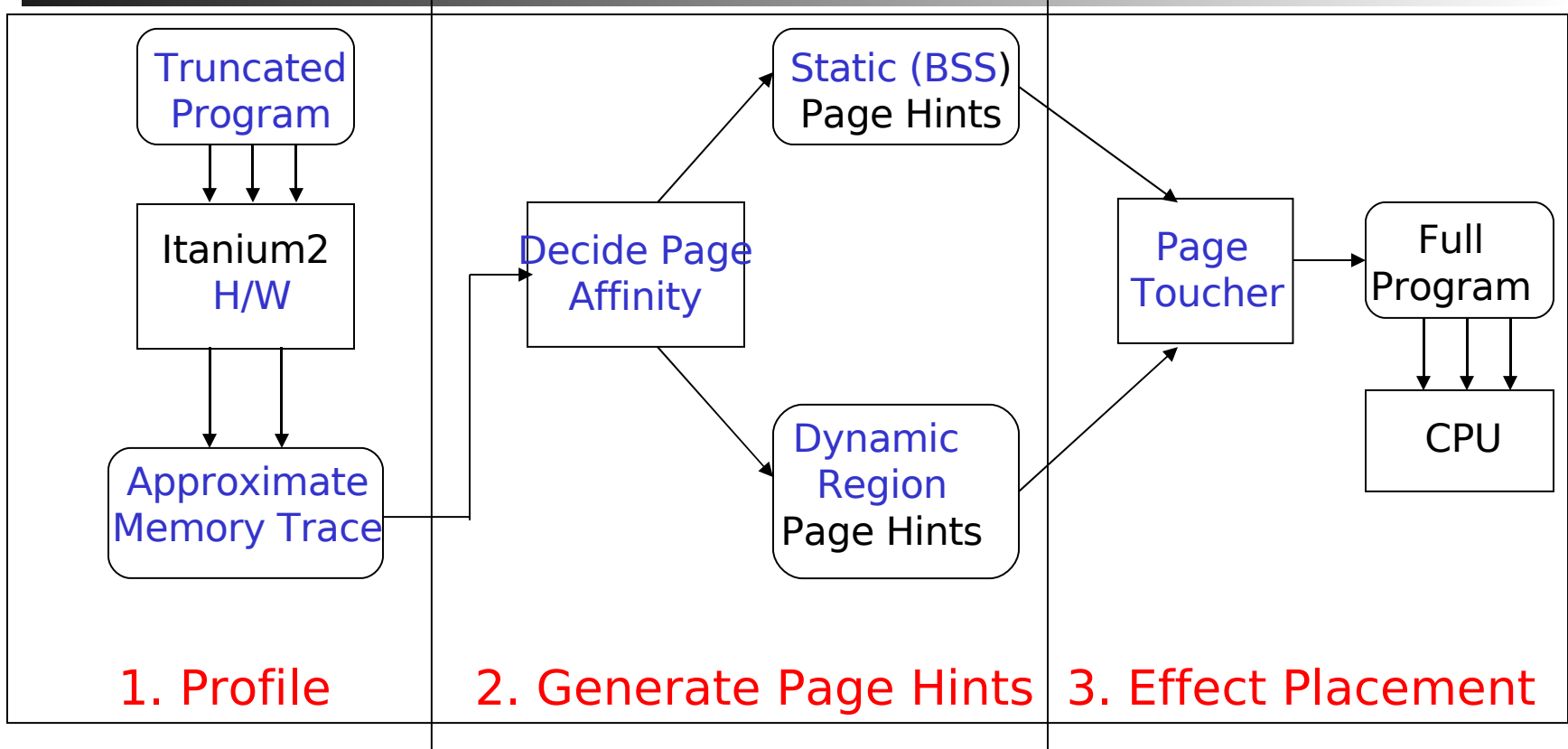


- **Problem:** Very hard for static compiler analysis to find dynamic access pattern of whole program.

- **Solution:** Use *memory trace profile* for allocation analysis.

First Processor-centric scheme, no n/w interconnect or compiler support required !

Our Approach

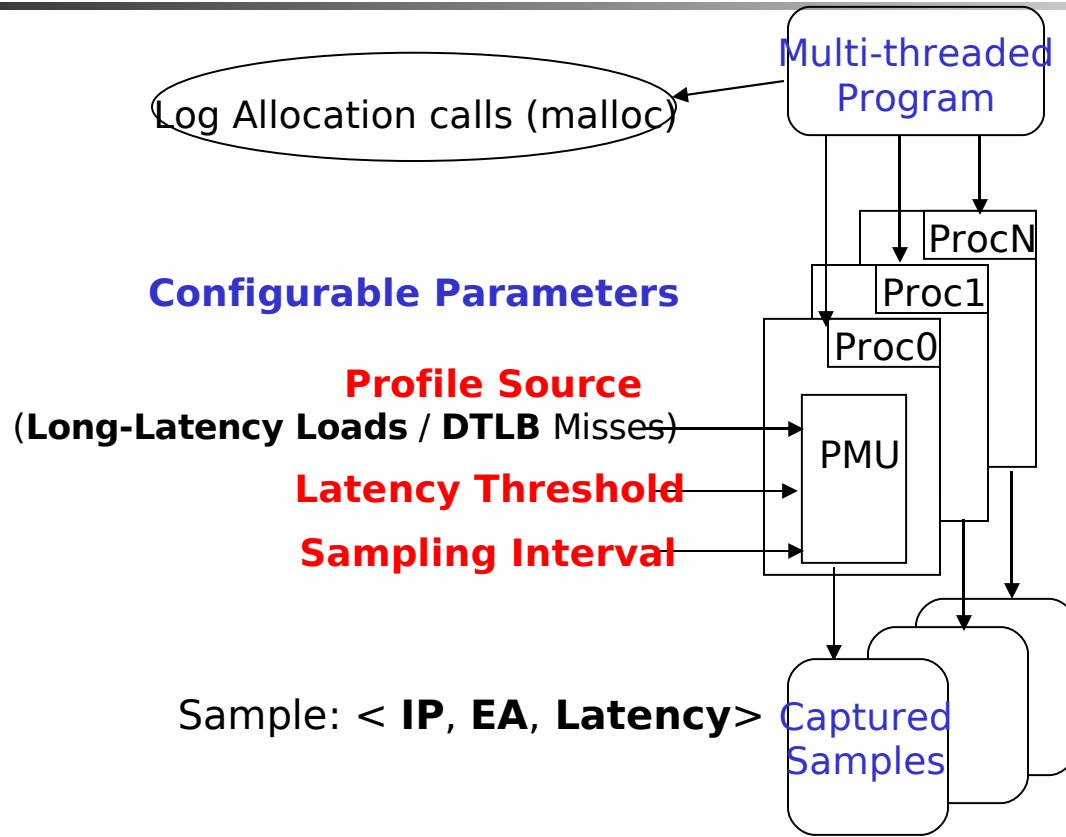


User marks **start, end of stable execution phase** (e.g., timestep)

1. **Profile** truncated program → **Approx. Memory Trace**
2. **Memory Trace** → **Page Affinity decision**.
3. **Run full program** → Use **First-touch** for page placement.

Simple, existing processor H/W, almost completely automated!

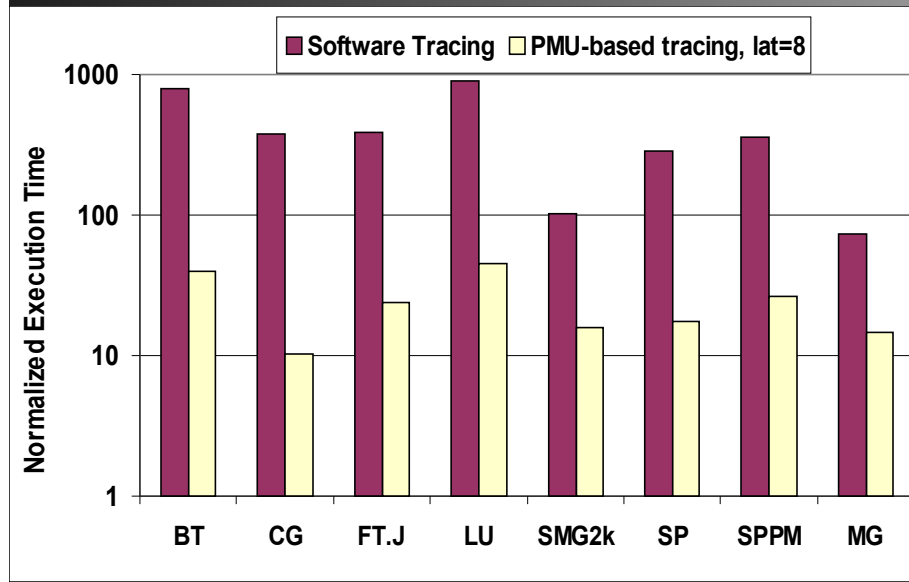
Phase 1: Profile Generation



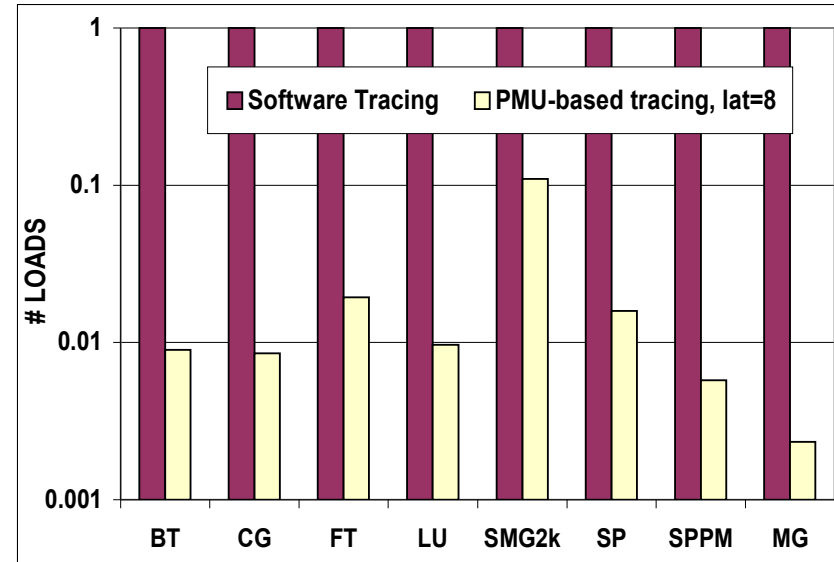
- **PMU** → Performance Monitoring Unit
 - Use **Perfmon2** interface to H/W counters.
 - **Latency Threshold** → Eliminate load hits in L1/L2/L3 caches
 - **Lossy Trace** (~90% loss), but that's OK for us.

PMU support perfect for capturing filtered approximate trace !₉

Key Point: Profile collection is *low-cost*



Execution Overhead (from ICS 05)

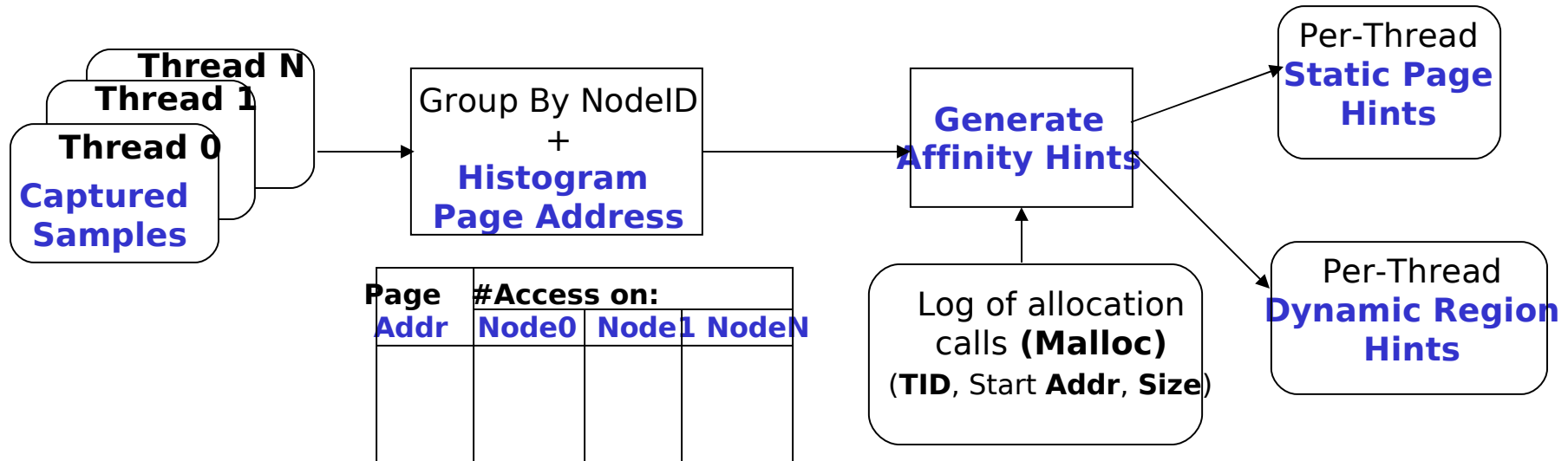


Samples (from ICS 05)

- Latency threshold removes most hits
- Lossiness & Sampling: Reduce trace size further.
 - At least 10X - 100X Faster than software tracing.
 - At least 10X - 100X Smaller trace size.
- No S/W modification required at all.
 - Binary rewriting, procedure cloning, etc.

PMU slashes overheads → Makes profile-based approach feasible !

Phase 2: Generate Page Affinity Hints



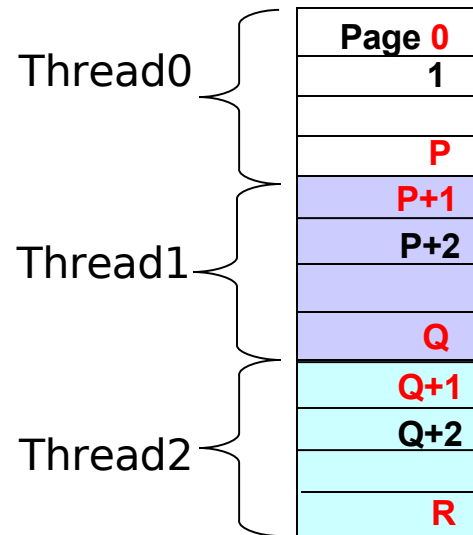
- Memory Access per Virtual Page → Node to map page on.
 - SGI Altix: 2 processors/node → Group accesses by node.
 - Histogram accesses to page by all nodes
 - Map page to node with most frequent access.
- For static (BSS) pages → per-thread page addresses.
- For dynamic regions (malloced) → calculate offset from region start

Example: Static Hints

```
//global array
double A[MATDIM];

//work
#pragma omp parallel for
schedule(static)
for(l=0;l < MATDIM;l++)
    B[l] = A[l]+A[l-1];
```

A: BSS Segment, Start=0x6000000000286760



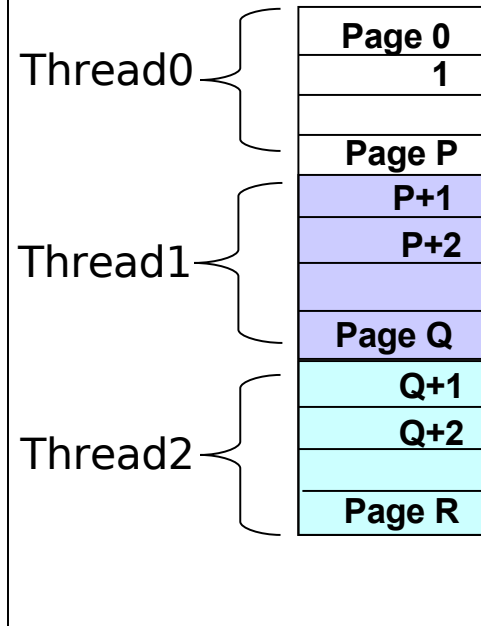
- 3 Threads, each accessing distinct "chunk" of array A.
- Static Affinity hints will be (system with 1 proc/node):
 - Thread0: Page 0, ..., P
 - Thread1: Page P+1, ..., Q
 - Thread2: Page Q+1, ..., R
- For static (BSS) pages → per-thread page addresses.

Example: Dynamic Region Hints

```
//pointer
double *A;

//only thread0 !
A = malloc( sizeof(double)*MATDIM);

//work
#pragma omp parallel for
schedule(static)
for(l=0;l < MATDIM;l++)
    B[l] = A[l]+A[l-1];
```



Dynamic Region Affinity:

Thread0: Malloc_Call_ID_0, Hints:

Node0: Byte Offset <0, PAGESIZE, 2*PAGESIZE,....>

Node1: Byte Offset < (P+1)*PAGESIZE,>

Node2:

Allocating Thread applies hints for region, after malloc returns !

Phase 3: Profile-guided Page Placement

- Run complete program → use affinity hints for page placement.
- “Touching” (load & store) virtual page from processor → Forces page allocation on that node.

Static Region (BSS) Hints

- Negligible overhead → Touching only once, at startup

Dynamically-allocated Region Hints

- Wrapper self-schedules thread on target node (`sched_setaffinity`)
- Touch byte offsets, context switch back to original node.
- User-Transparent, but expensive
- Reduce overhead by parallelizing first-touch → Future Work

Benefit mostly overcomes overhead → Good wallclock improvements !

Experimental Setup

Benchmarks

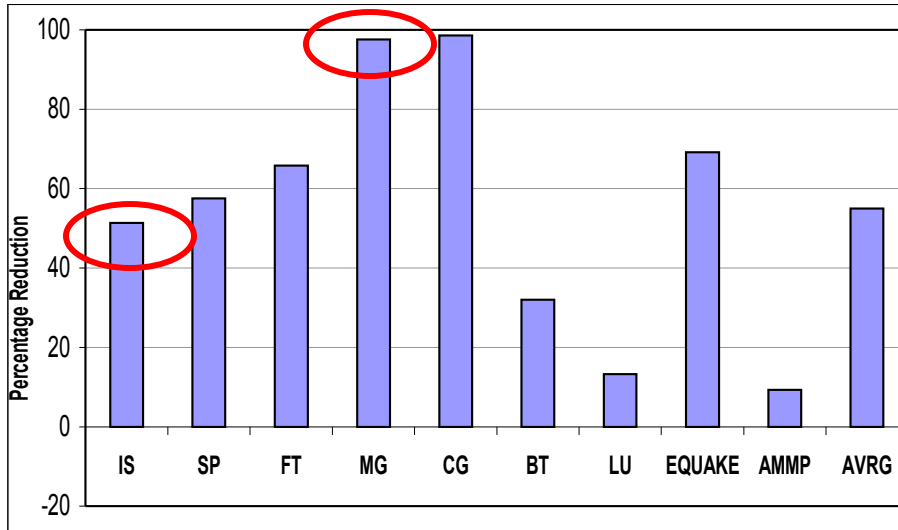
- Scientific Workloads
- NAS-2.3 C benchmarks (OMNI Group) : IS, SP, FT, MG, CG, FT, LU
- SPEC OpenMP 2001: 320.equake, 332.amp
- 5 benchmarks have static global arrays, 4 have dynamic allocation.
- Largest data sets (NAS: Class C), SPEC: Ref data set.

Platform

- SGI Altix, ccNUMA, Itanium2 processors, 2 procs/node.
- Used 8 processors.
- 1 OpenMP thread → 1 processor (bound threads).
- All measurements on non-interactive nodes

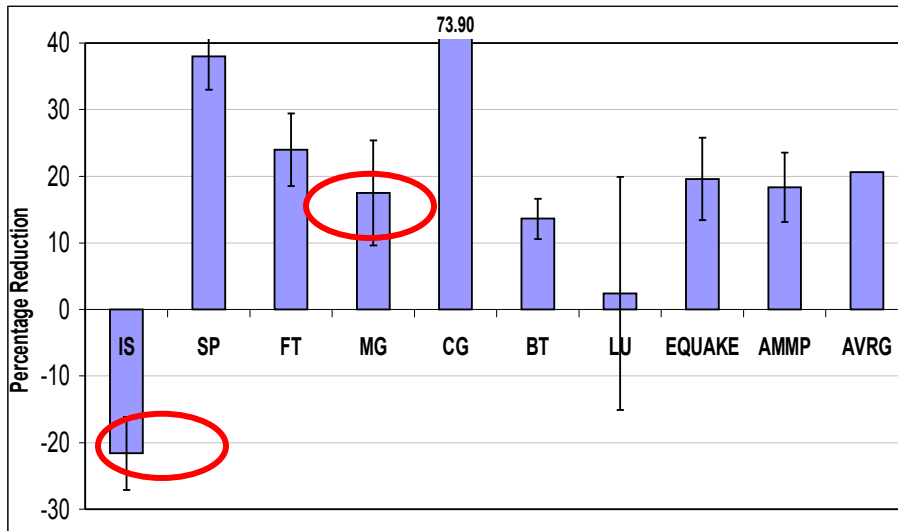
All Phases (Profiling/Analysis/Placement) **parallelizable**
Should scale easily to larger #procs !

Performance Improvements



% Reduction in Remote Loads

- Original VS Profile-guided
- **54% reduction on average**

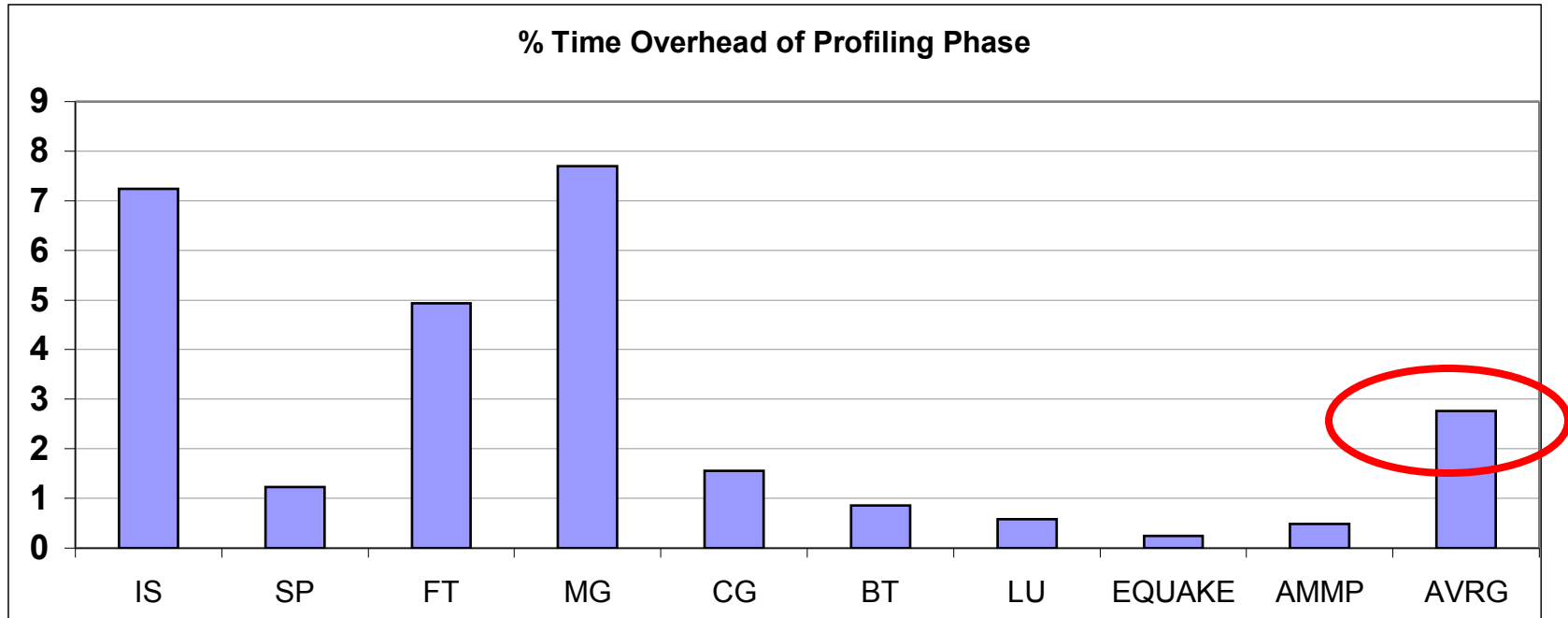


% Reduction in Wallclock time

- Error bars: 95% Conf. Interval
- **20% improvement on average**
- CG has 73% improvement !
- **MG, IS → Could be better**
- Includes "Touching" Overhead

Good improvements with minimal user effort !

Profile Cost: % Time Overhead



$$\text{Overhead} = \frac{\text{Time for profiling truncated program}}{\text{Time for complete original program}}$$

- Profile time overhead = 2.7% of COMPLETE original program time.
- Time for "single timestep" high, but need only one timestep.

Decreasing Sampling Intervals

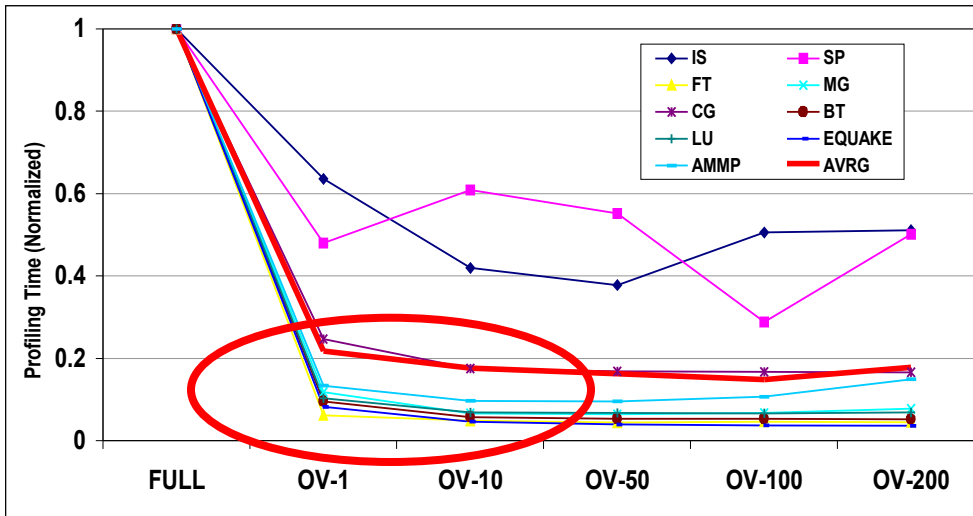
- H/W can vary sampling interval
- **Tradeoff:** Sampling Intervals \leftrightarrow Profile Cost, Quality ?
- Measure Cost, Quality against “Maximum information profile” (MIP).

MIP = Lowest sampling interval (1),
Lowest latency threshold (4 cycles)

- Compare MIP against:
 - Fixed latency threshold = 128 cycles (> L3 hit latency)
 - Different Sampling intervals: OV1, OV10, OV20, OV100, OV200

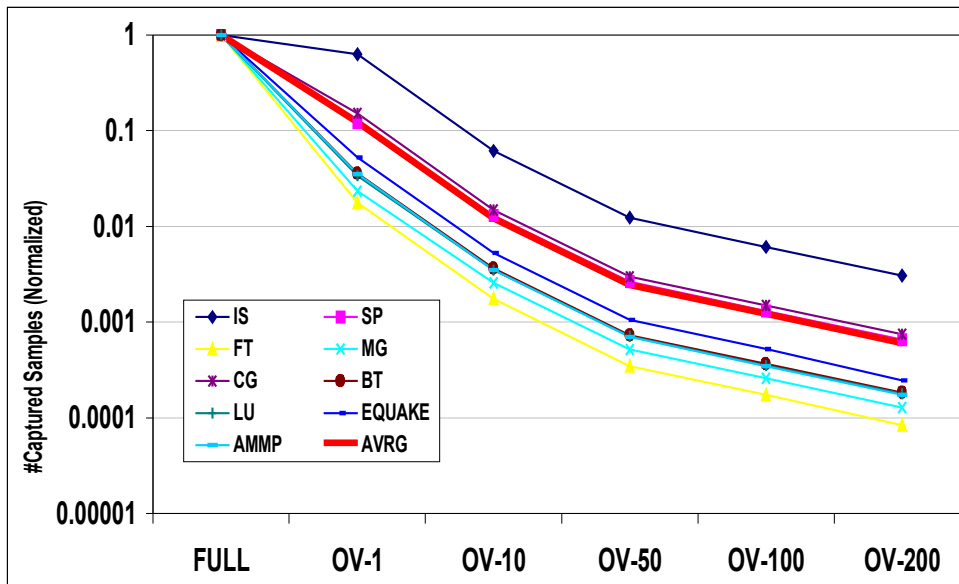
Q: ⑧ Sampling Interval \rightarrow Less Info
 \rightarrow Impact on Profile Cost, Coverage, Accuracy ?

Profile Cost & Size



Profiling Overhead

- FULL: Max. Info. Profile
- Only 20% of FULL at OV1
- **OV1 or OV10 sweet spot.**



Samples in Trace

- Approx. **linear decrease**
- As expected.

Profile Quality: Coverage & Accuracy

How does ⑧ sampling intervals affect profile quality ?

- Use Max. Info. Profile (MIP) as reference

Coverage

- Fraction of Pages that we have affinity hints for.
- Low Coverage → Insufficient number of hints !

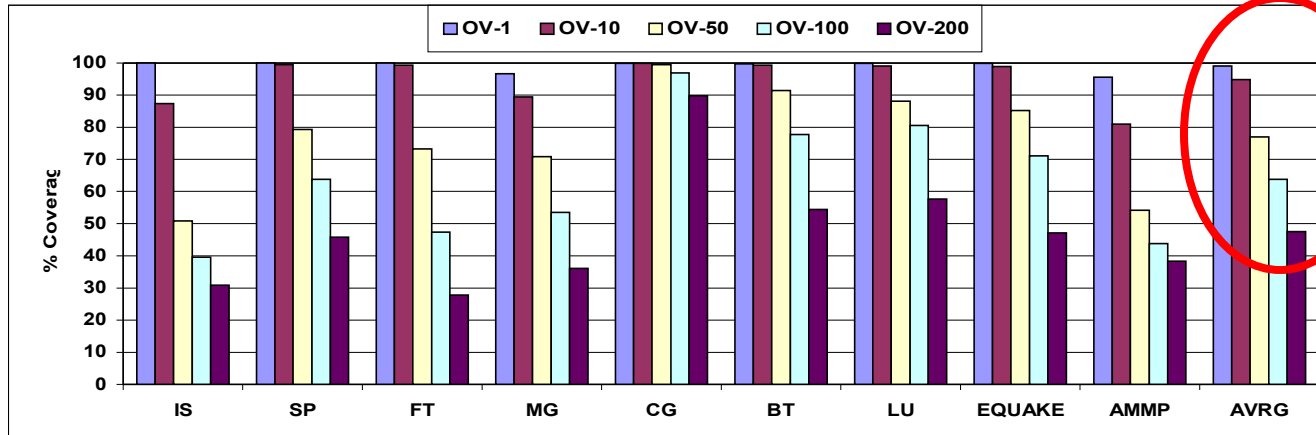
Accuracy

- Fraction of Pages with MATCHING affinity values (to MIP)
- Low Accuracy → Hints do not match (misleading profile)
- High Accuracy → Same affinity hints as MIP.

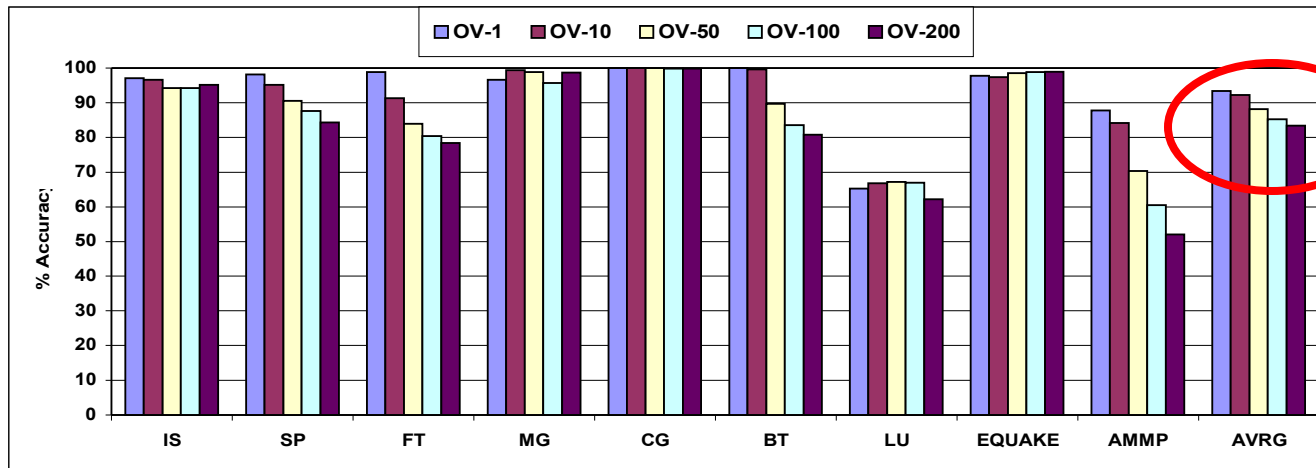
Ideally, want 100% Accuracy And High Coverage.

Profile Quality: Coverage & Accuracy

Coverage



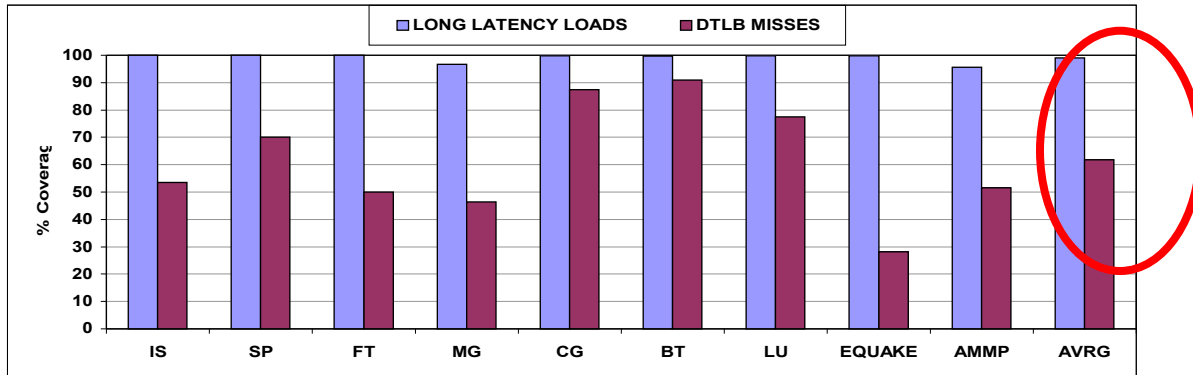
Accuracy



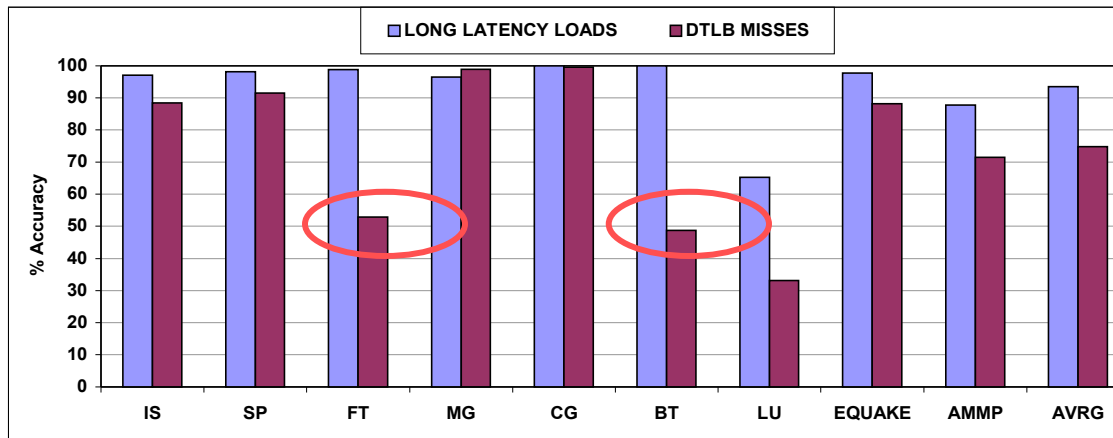
- Coverage Falls, but Accuracy remains high.
- Same affinity decision as with Max. Info. Profile (懺 Accuracy)
- But low number of hints (悪 Coverage) ↘ less impact.

Using TLB misses as profile source

Coverage

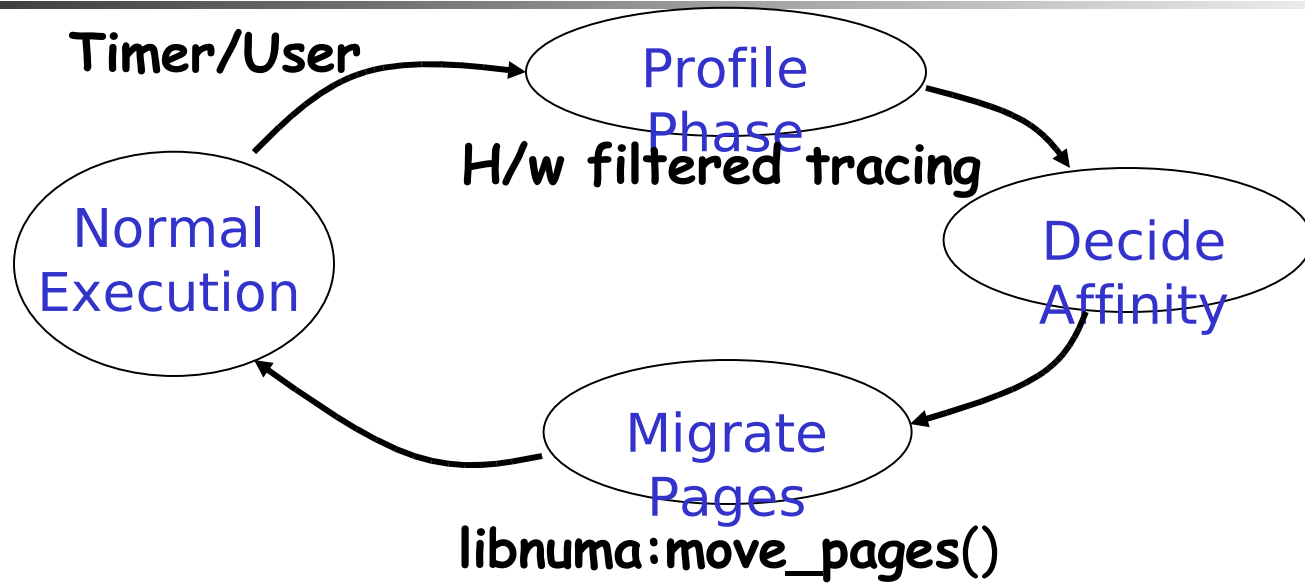


Accuracy



- Long Latency Loads vs. DTLB Misses (sampling interval=1)
- DTLB always lower coverage, sometimes lower accuracy too.
- **DTLB Misses not a good indicator of page accesses.**
 - Eg, Good TLB locality, but poor cache locality.

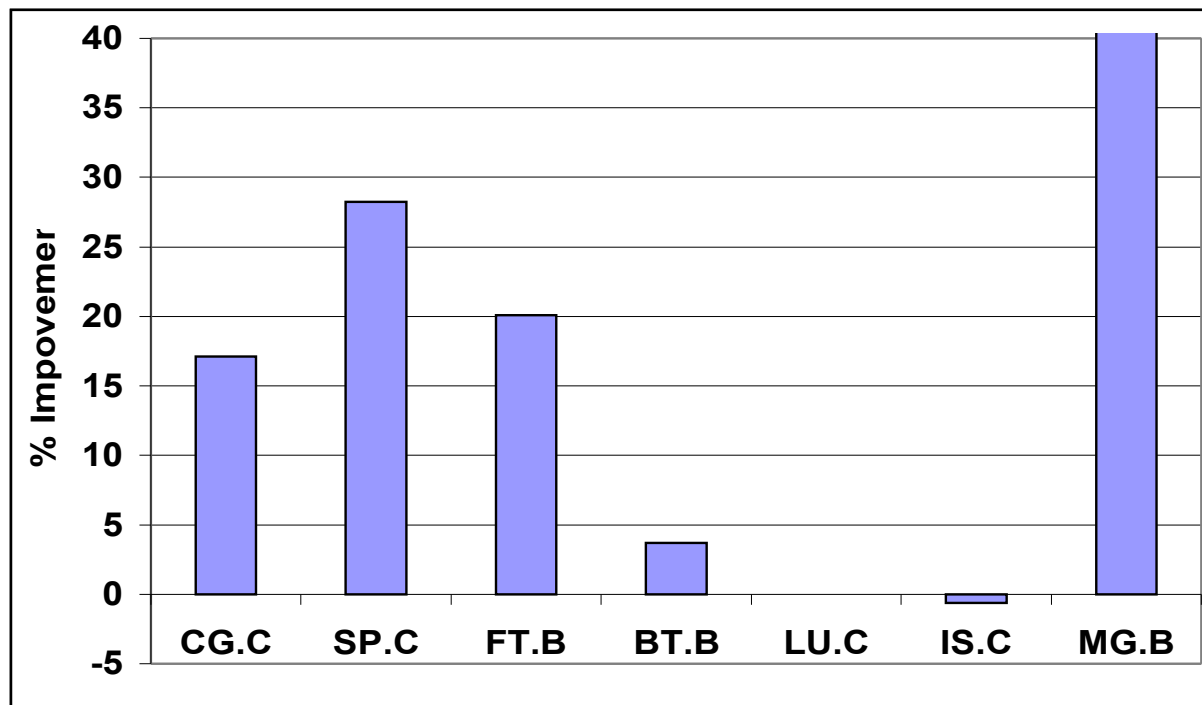
Current Work



- **Past Work** ➤ System supported only “first-touch” placement (2.4 kernels)
- **Ongoing Work** → Use new **libnuma library API** (2.6+ kernels)
 - Allows **user-directed dynamic page migration**.
 - **No need for profile run**.
 - Libnuma functions to bind threads + memory.

Other Platforms

- H/W tracing available on other platforms
 - Intel Xeon/Pentium4, IBM Power5, AMD K10(?)
- Our Framework is *portable* - uses only processor-centric H/W.
- **Early results on 4-node AMD Opteron ccNUMA:**



**Significant
Wall-clock time
Improvement**

- **Proof of concept:** L2 miss trace on Xeon (PEBS@Perfmon2) ↘ Use for page placement on the Opteron.

Related Work

“Using hardware counters to automatically improve memory performance” by M. Tikir and J. Hollingsworth, SC 2004

- + Sample bus transactions ↘ migrate pages on affinity.
- Need special h/w for bus transaction sampling.
- Shared hardware ↘ Only one app can do profiling at a time.
- Problems with dynamically allocation?

“UPMLIB: A run-time system for tuning the memory performance of openmp programs on scalable shared memory processors” by D. Nikolopoulos et. al., 2000

- + Measure remote & local requests to a page.
- + Use this info to migrate pages
- Compiler support to mark candidate arrays
- H/w support: per-page hardware counters in each node
- OS support: IRIX Memory Locality Domains, proc mapping.

Highlights & Ongoing Work

Low-cost, Automated page-placement scheme


- + *Processor-Centric Approach.*
- + *No Compiler or Network interconnect support required.*
- + *Transparently handle both static (BSS) and dynamic (malloced) regions.*
- + *Simple H/W support, also available in Pentium4 and POWER5*
- + *Good Wallclock savings, minimal user effort.*

Ongoing Work

- *User-level dynamic page migration with new **libnuma API.***
 - + *No separate profile run.*
 - + *Handle programs with *changing* memory access patterns.*



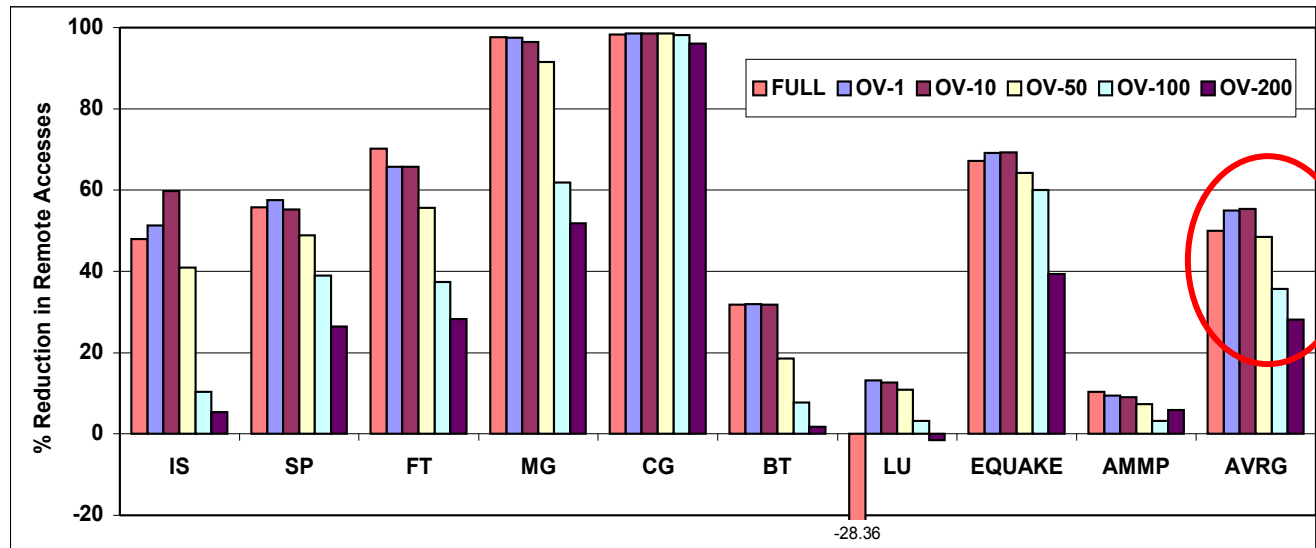
The End



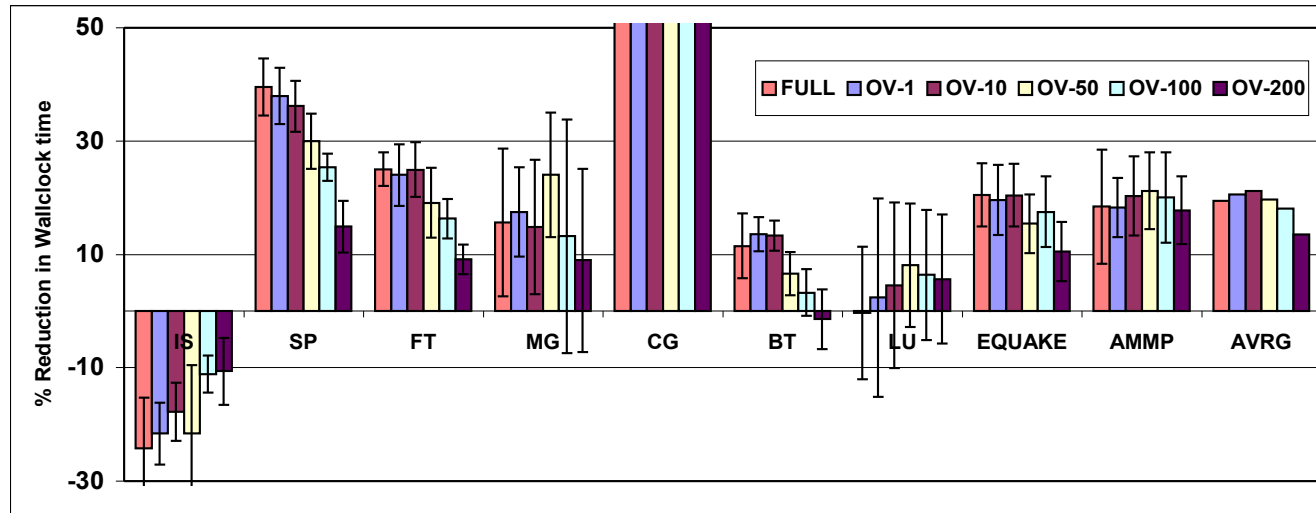
Backup Slides

Long_Lat : Reduction in Execution Time & # Remote Loads

% Reduction in Remote Loads

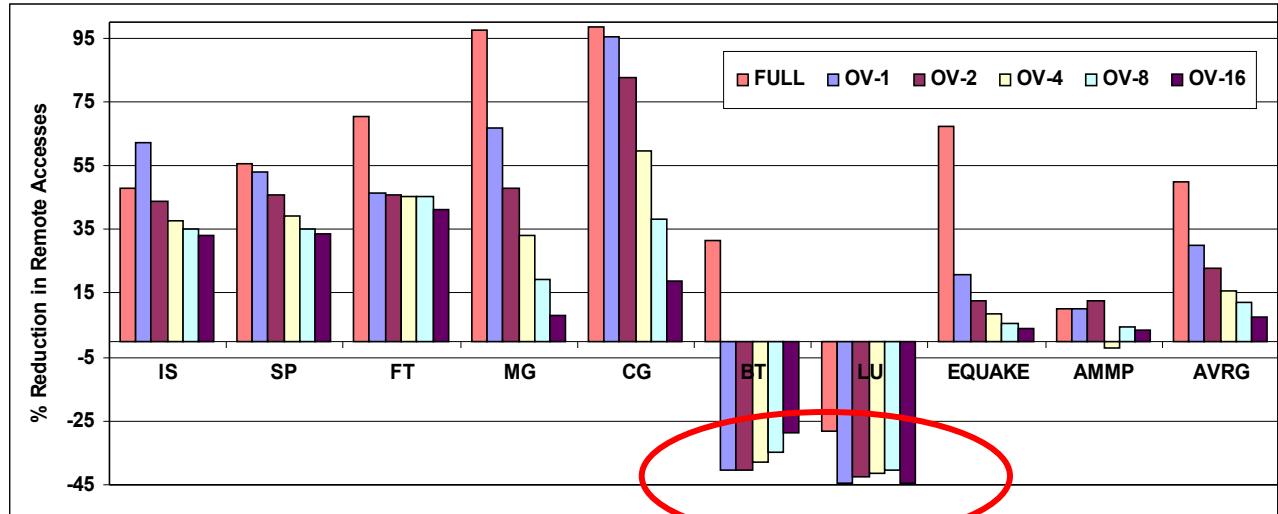


% Reduction in Wallclock Time



DTLB : Reduction in Execution Time & # Remote Loads

% Reduction in Remote Loads



% Reduction in Wallclock Time

