

# Hardware Profile-guided Automatic Page Placement on ccNUMA systems

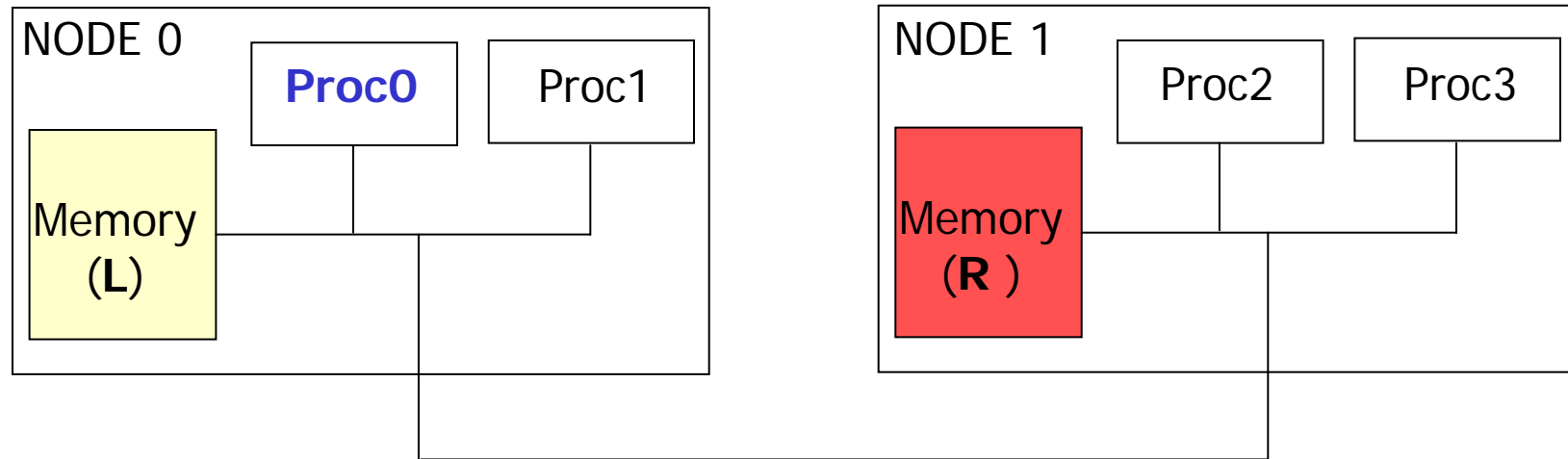
Jaydeep Marathe, Frank Mueller



North Carolina State University, Dept. of CS

# The Opportunity

NUMA = *Non-Uniform* Memory Architecture



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

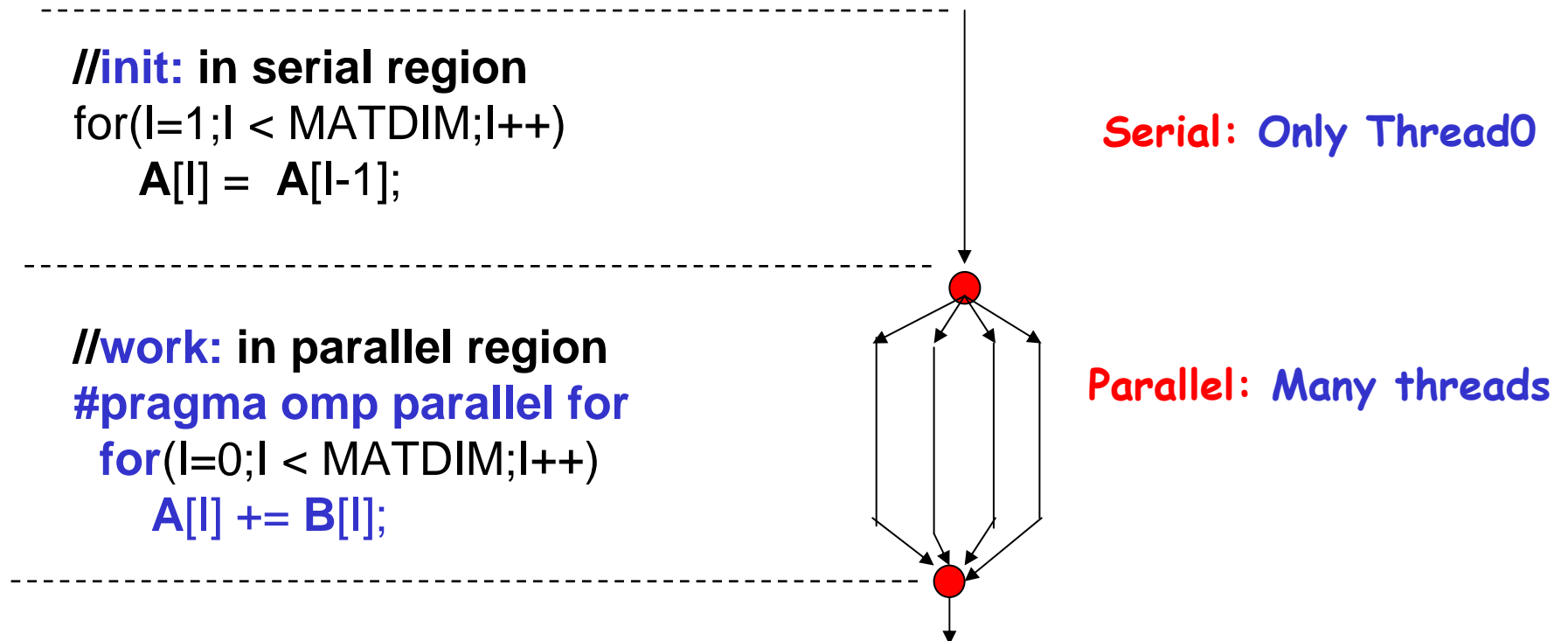
Proc0 → Local Memory L : 207 cycles

Proc0 → Remote Memory R : 409 cycles

Allocating Memory Near Computation → **Big Performance Impact.**

# Our Environment

- SGI Altix ccNUMA system, Itanium2 processors.
- Node: 2 Processors + Physical Memory.
- *Bound OpenMP threads.* (Thread0: Proc0, ...)
- OS (Linux) only supports *First-Touch Page Allocation.*



OpenMP's **fork-join** parallelization model

# Example: Subtle Page Placement Badness

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

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

```
//Hot Function: > 90% exec time
SMVP()
{
  #pragma omp parallel
  {
    ... = ARCHmatrixindex[];
    ... = ARCHmatrixcol[];
  }
}
```

**Remote Accesses !!**

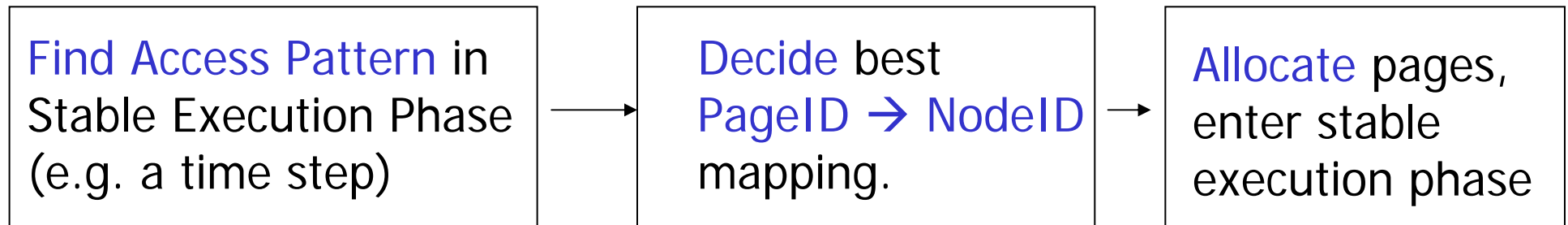
- Memory initialized from file → Serialized.
  - ARCHmatrixindex[] and ARCHmatrixcol[] allocated on node0.
- 21% increase in wallclock time.

**Sometimes initialization is *inherently* serial !**

# Achieving better page placement

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

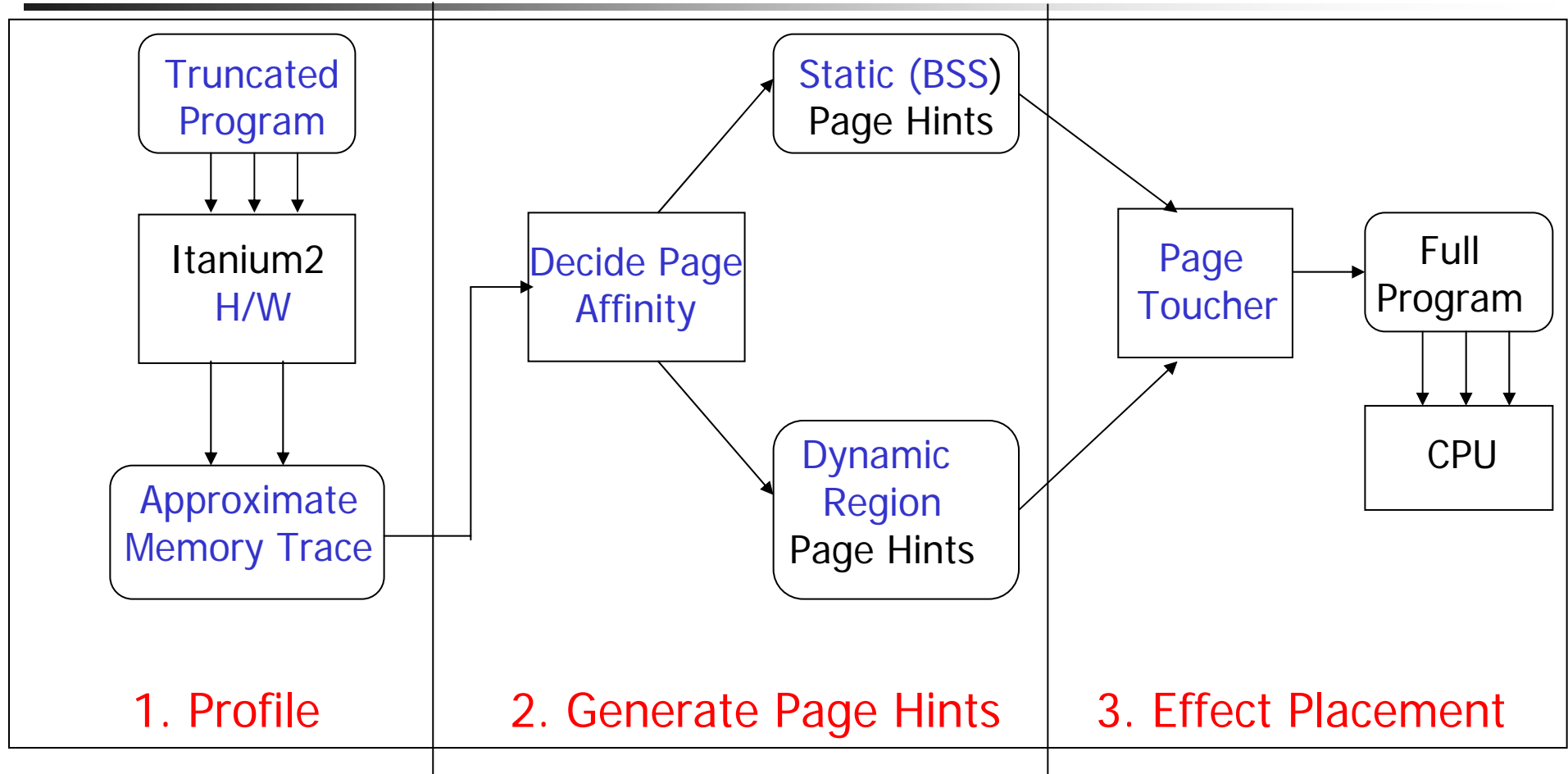
Ideally:



- **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

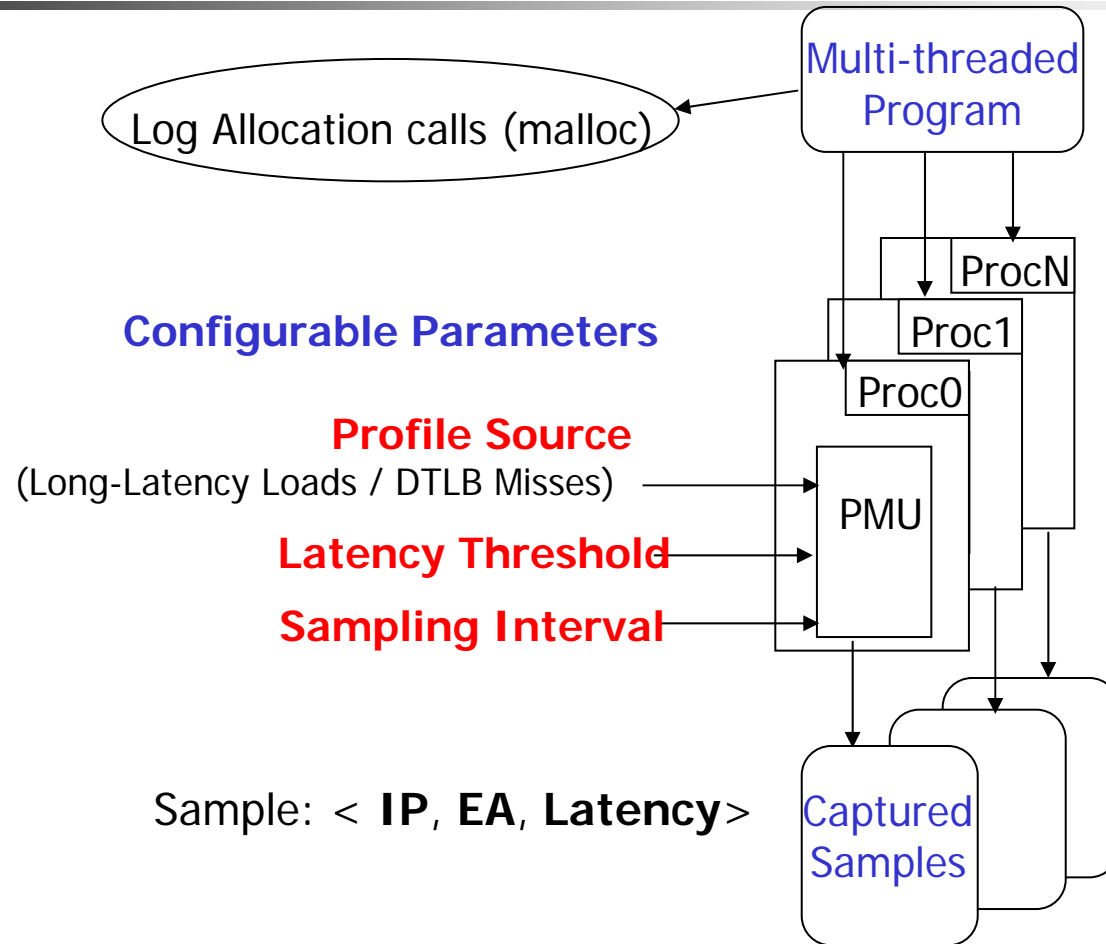


User marks **start, end of stable execution phase** (e.g., timestep)

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 !**

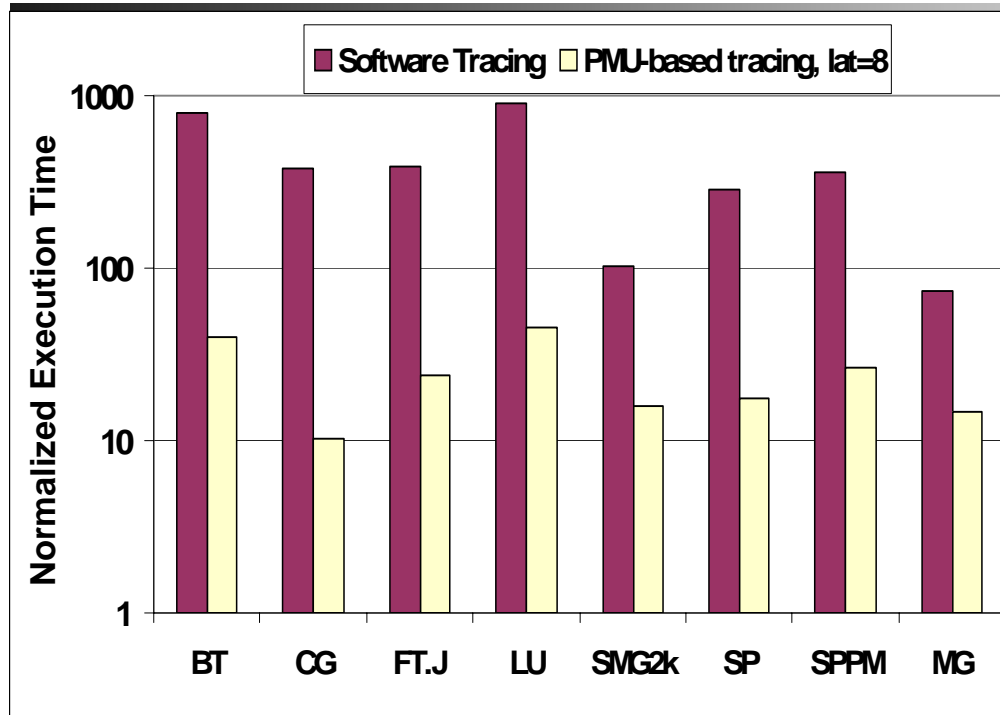
# Profile Generation



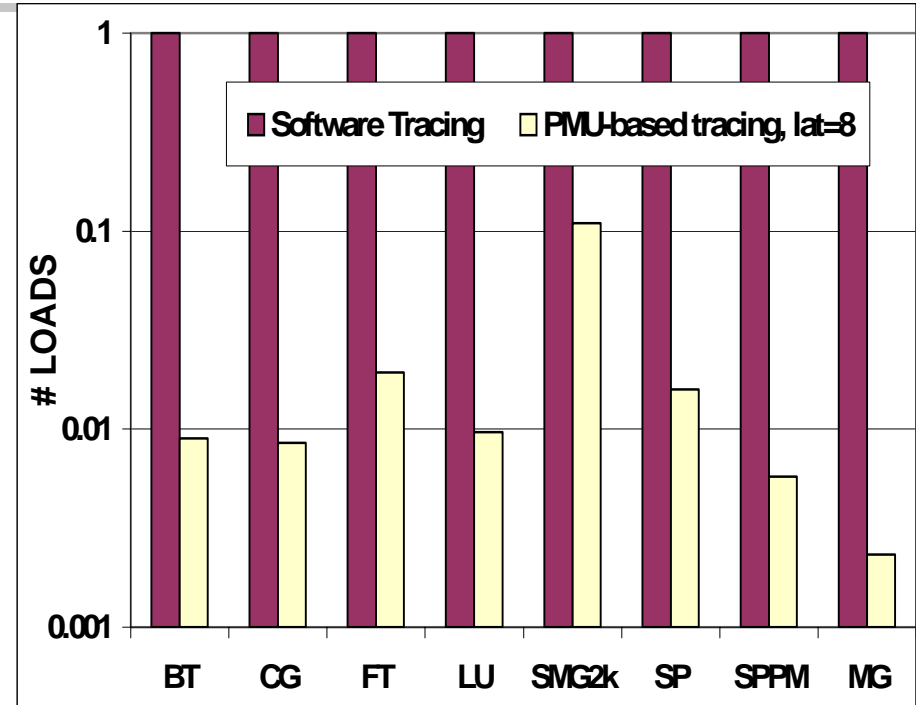
- **PMU** → Performance Monitoring Unit
  - Capture Long-Latency Loads / DTLB Misses
  - Latency Threshold → Eliminate load hits in L1/L2/L3 caches
  - Lossy Trace (~90% loss), but that's OK for us.

**PMU support perfect for capturing filtered approximate trace !**

# Key Point: Profile collection is *low-cost*



Execution Overhead (from ICS 05)

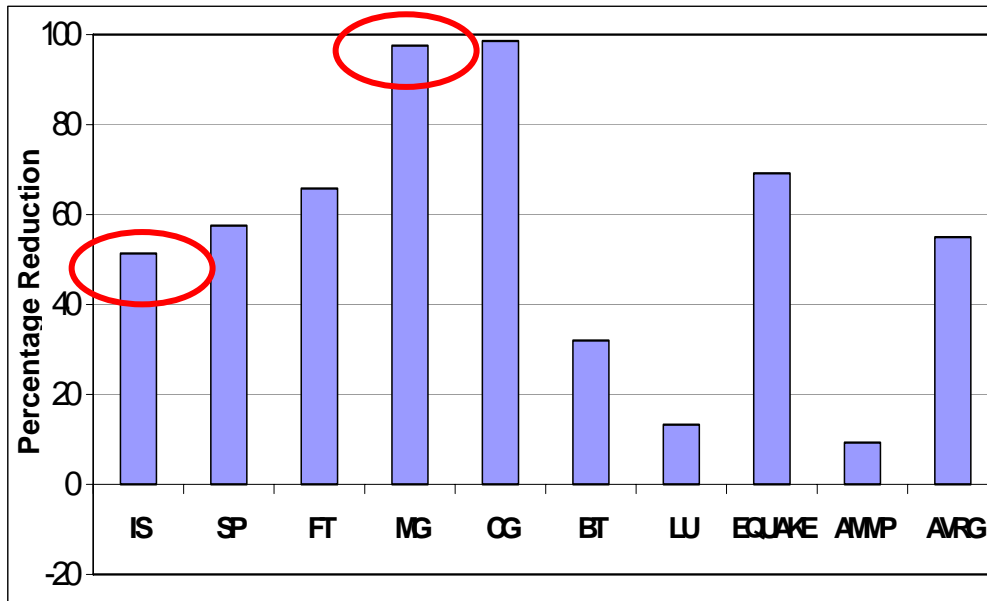


# Samples (from ICS 05)

- **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.

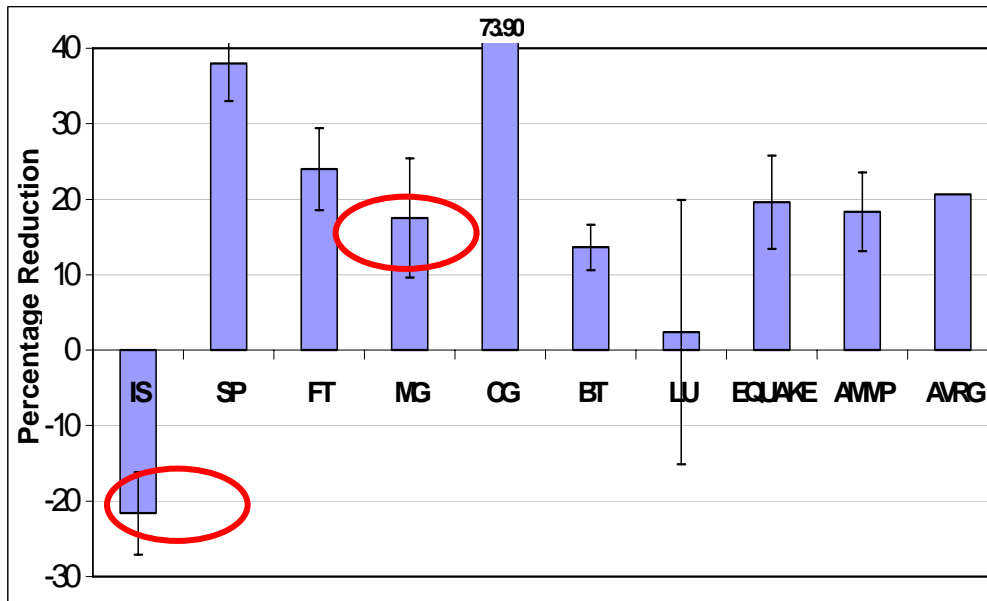
**PMU slashes overheads → Makes profile-based approach feasible !**

# Performance: Un-tuned NAS-2.3-C & SPEC 2001-OMP programs



## % 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 !**