Automatic Compiler-Inserted I/O Prefetching for Out-of-Core Applications

Todd C. Mowry

Angela K. Demke

Orran Krieger

Department of Computer Science

Department of Electrical and Computer Engineering

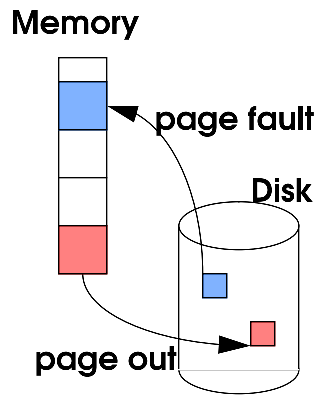
University of Toronto

Numeric Out-of-Core Applications

- **Large class of important scientific problems**
  - Examples: visualization, weather modeling, fluid dynamics..
- **Inherently large data requirements**
- **Memory acts as a cache of data that resides on disk**
- **I/O not fundamental part of problem being solved**

☞ **Natural to scale programs using paged virtual memory**

Problems with Paged Virtual Memory

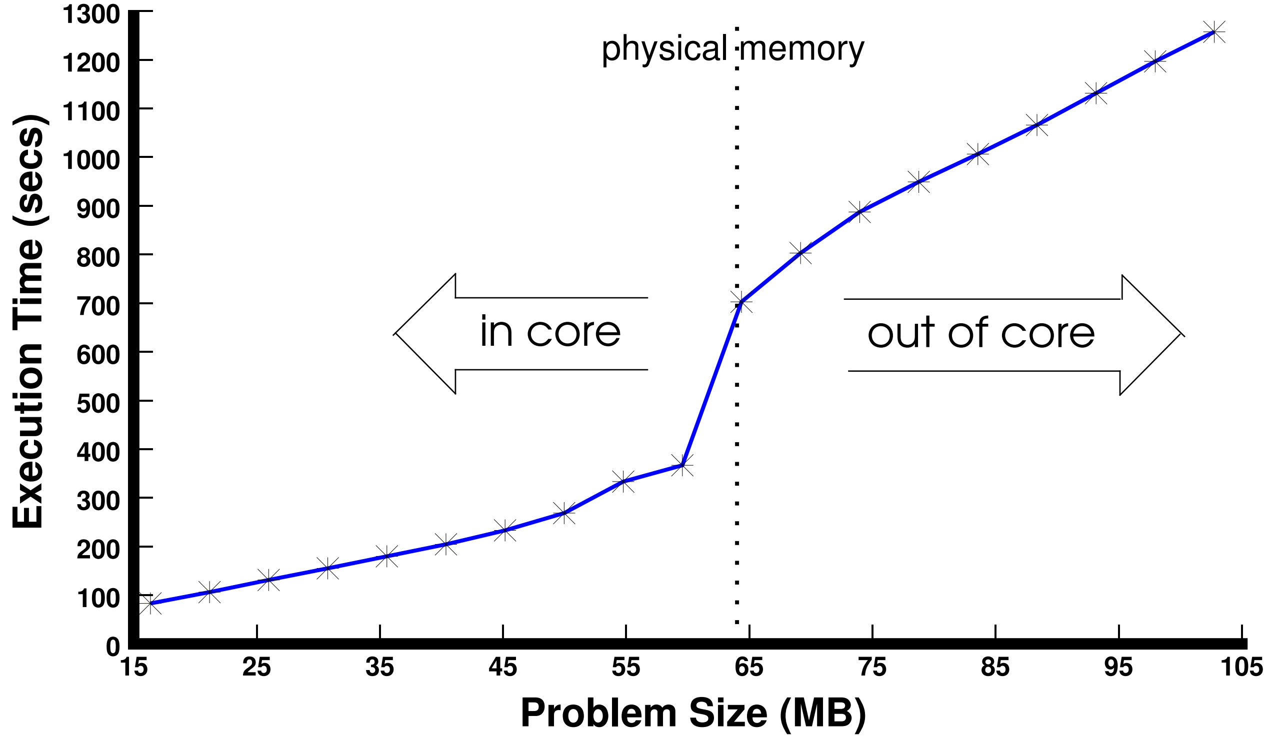


- disk reads triggered by page faults incur full latency
- small requests cannot exploit available bandwidth
- prefetching limited to simple access patterns
- memory manager may make poor replacement choice

☞ not viable solution due to poor performance

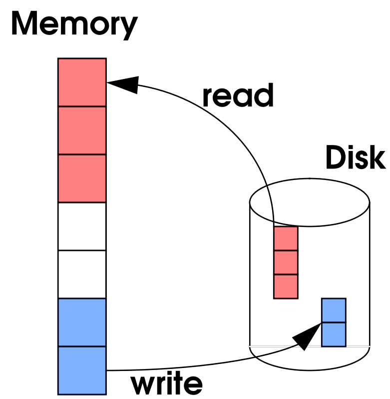
An Example: BUK

- computation grows linearly with problem size



☞ Performance suffers when data set > available memory

Problems with Explicit I/O



- substantial programmer effort required
- may negatively affect numerical stability of algorithm
- low-level policy decisions → non-portable solutions

☞ **A better approach is needed!**

Goals:

- **Ease of use**
  - maintain abstraction of unlimited virtual memory
- **High performance**
  - exploit available resources
  - hide I/O latency
- **Minimize Operating System Support**
  - easy to incorporate into existing systems
  - avoid adding overhead to common path

☞ **Automatic compiler-inserted I/O prefetching**

Keys to Success

- **Exploit available I/O bandwidth**
  - achieved via multiple outstanding requests
- **Separate request for data from use of data**
  - re-structure code and insert *prefetch* operations
- **Avoid displacing useful data from memory**
  - identify good candidates for replacement via *release*

Overview

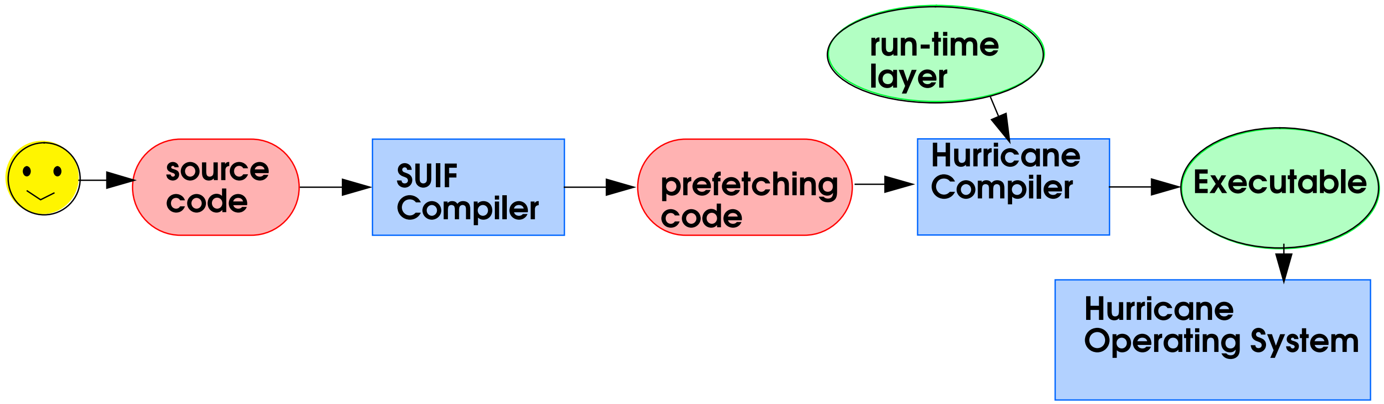
☞ **Implementing Automatic I/O Prefetching**
  - Compiler analysis
  - Operating system support
  - The run-time layer support

- **Experimental Results**

- **Conclusions**

System Components

- Programmer uses virtual memory abstraction, ignores I/O
- Compiler analyzes program, inserts *prefetch* & *release*
- Operating system supports basic primitives
- Run-time layer adapts to dynamic behavior to reduce overhead



Compiler Analysis

- **use extension of cache prefetching algorithm**
  - locality analysis is used to predict misses
  - loop splitting techniques isolate faulting references
  - software pipelining schedules prefetches ahead of uses
  - prefetches dense and *indirect* references
- **many similarities between cache and I/O prefetching**

	Cache Prefetching	I/O Prefetching
“Cache” Size	32kB	64MB
“Line” Size	32B	4kB
“Miss” Latency	100 cycles	100,000 cycles

- **to first order, change parameters accordingly**

☞ **However, there were some significant differences**

Extensions to Compiler Algorithm

Spatial Locality:

- split loops by strip-mining rather than unrolling
- amortize overhead & exploit bandwidth by block requests

Release Operations:

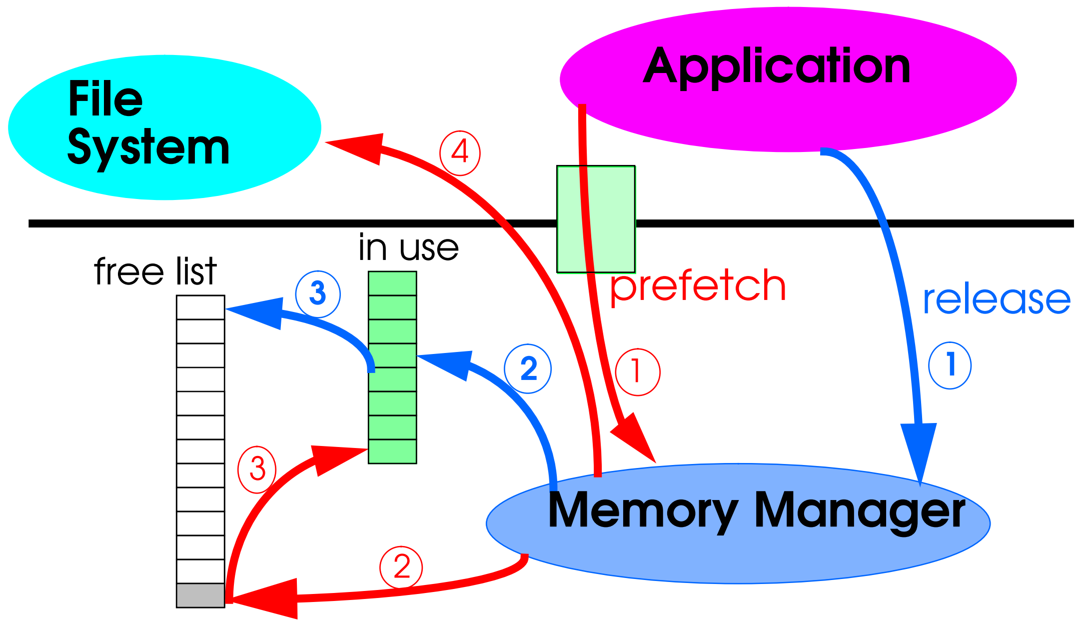
- identify pages that will not be used again soon
- finding "trailing" reference is straightforward

Small Loop Bounds:

- inner loop nests may access < 1 page of data
- must software pipeline across appropriate loop nest
- use type information & aggressive constant propagation

Operating System Support

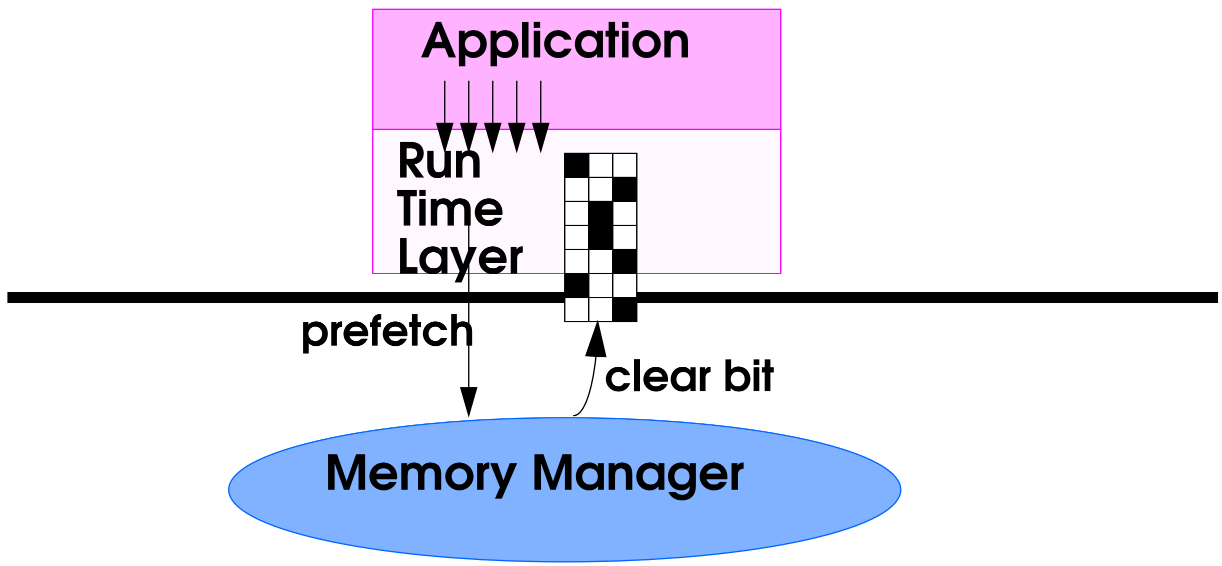
- **minimal support provided for basic operations**
  - *prefetch*: allocate page and send request to file system
  - *release*: move specified page to end of free list



- **provides map of pages in memory to user level via shared page**

Run Time Layer Support

- **reduces overhead by adapting to dynamic behavior**
  - uses map of pages in memory provided by OS to *filter* unnecessary prefetches



☞ **Key to obtaining good performance!**

Overview

✔ **Implementing Automatic I/O Prefetching**

☞ **Experimental Results**

- overall performance results
- importance of the run-time layer
- effects on system resources
- problem size variations

- **Conclusions**

Experimental Framework

Benchmarks

- NAS Parallel benchmark suite
  - eight numeric applications, scaled to out-of-core sizes
  - exhibit sequential, random, and strided access patterns

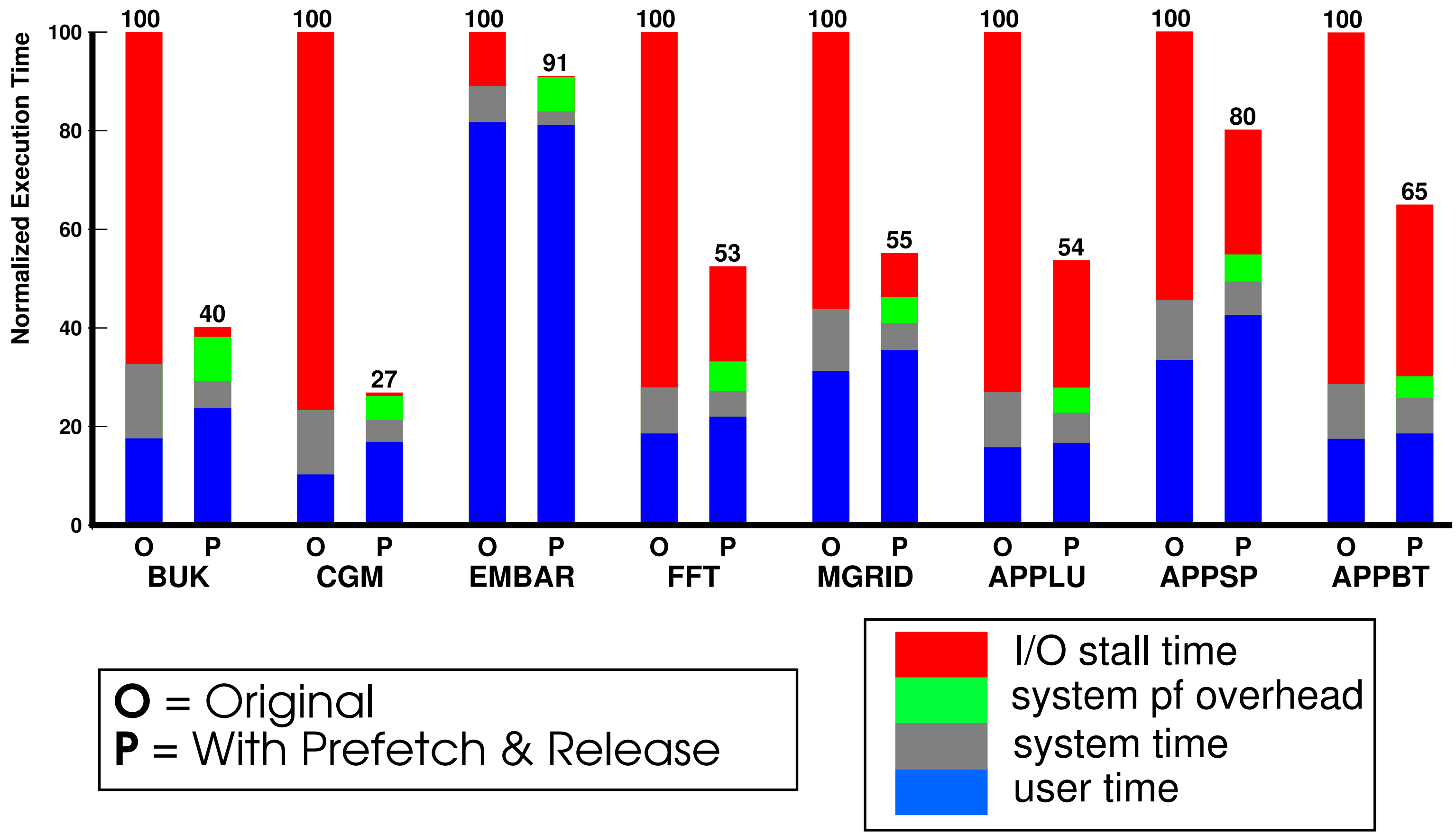
Hardware

- Hector multiprocessor
- array of 7 disks

Software

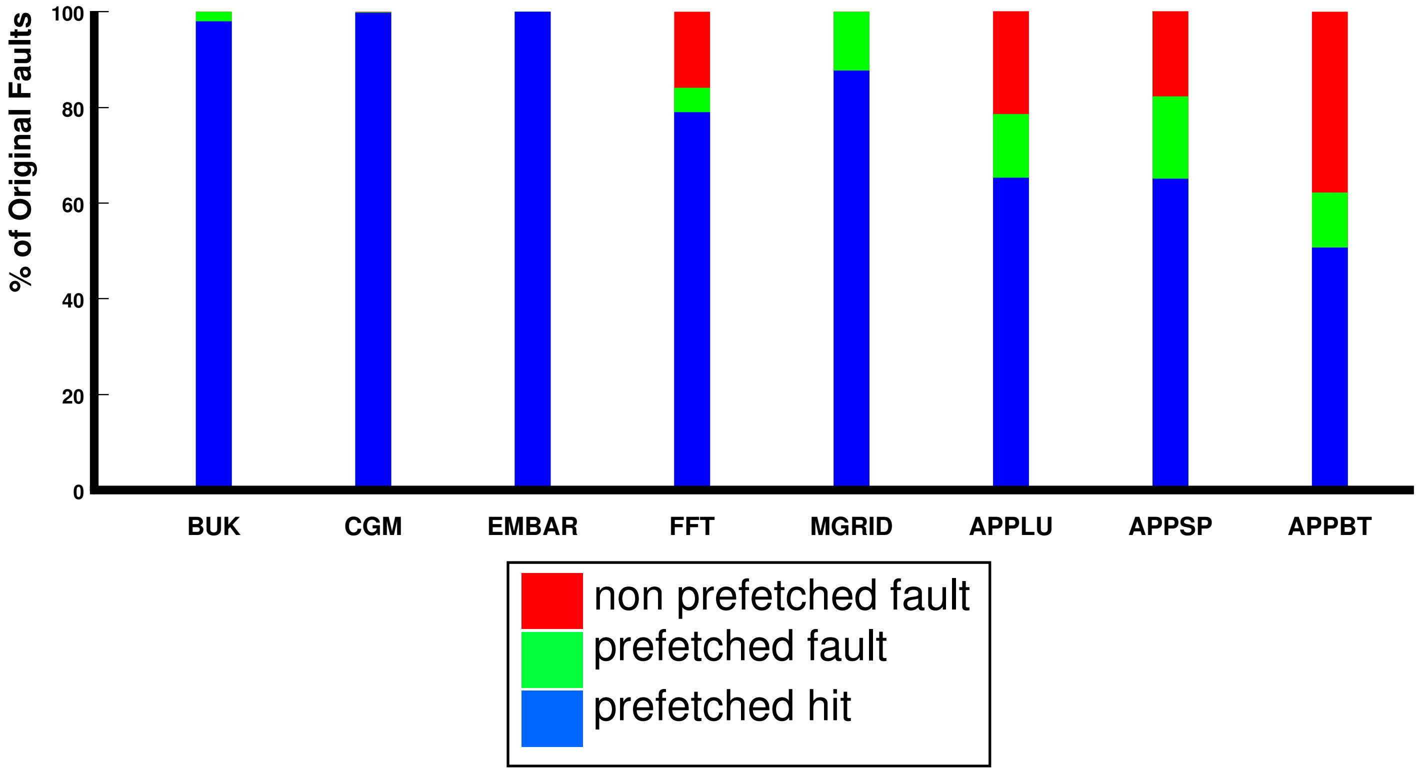
- Hurricane Operating System
- Hurricane File System
- SUIF Optimizing Compiler

Overall Performance Results



- more than half of the I/O stall time is eliminated in all cases
- over half the applications achieve nearly twofold speedup

Coverage Factor



- coverage factor = prefetched hit + prefetched fault
- 7 out of 8 have coverage factors greater than 75%
- small *prefetched fault* fractions => effective scheduling

University of Toronto

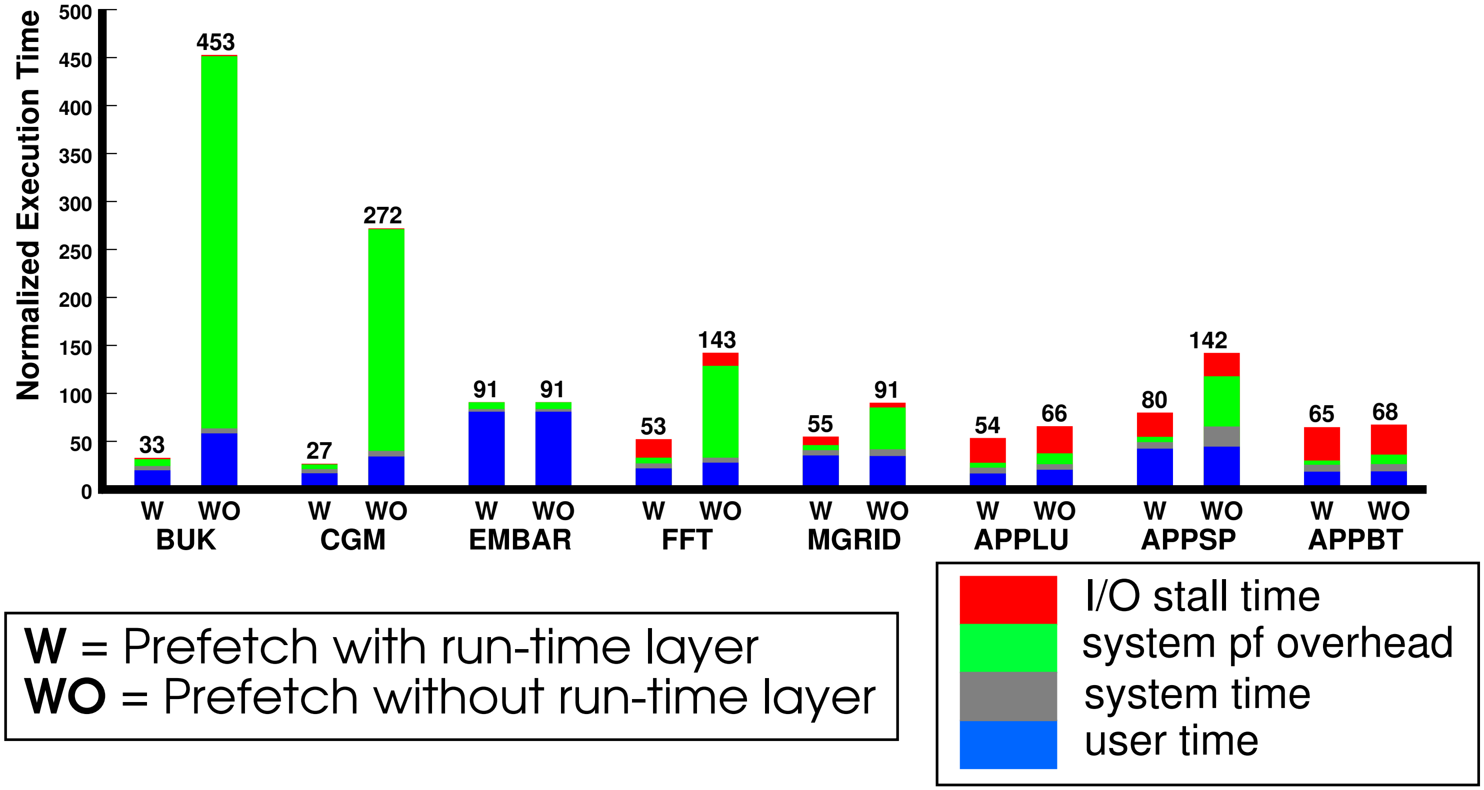
Unnecessary Prefetches

Benchmark	Inserted Prefetches Filtered at Run-Time	Unnecessary Prefetches Issued to OS
BUK	99.79%	0.07%
CGM	99.74%	0.08%
EMBAR	0.02%	0.00%
FFTPDE	99.59%	7.99%
MGRID	99.17%	8.03%
APPLU	96.99%	3.75%
APPSP	99.51%	7.55%
APPBT	98.31%	2.54%

- run-time layer removes many scheduled prefetches
  - >95% for all but EMBAR
- very few unnecessary prefetches are sent to OS

☞ **run-time layer is successful at filtering prefetches**

Performance Without Run-Time Layer



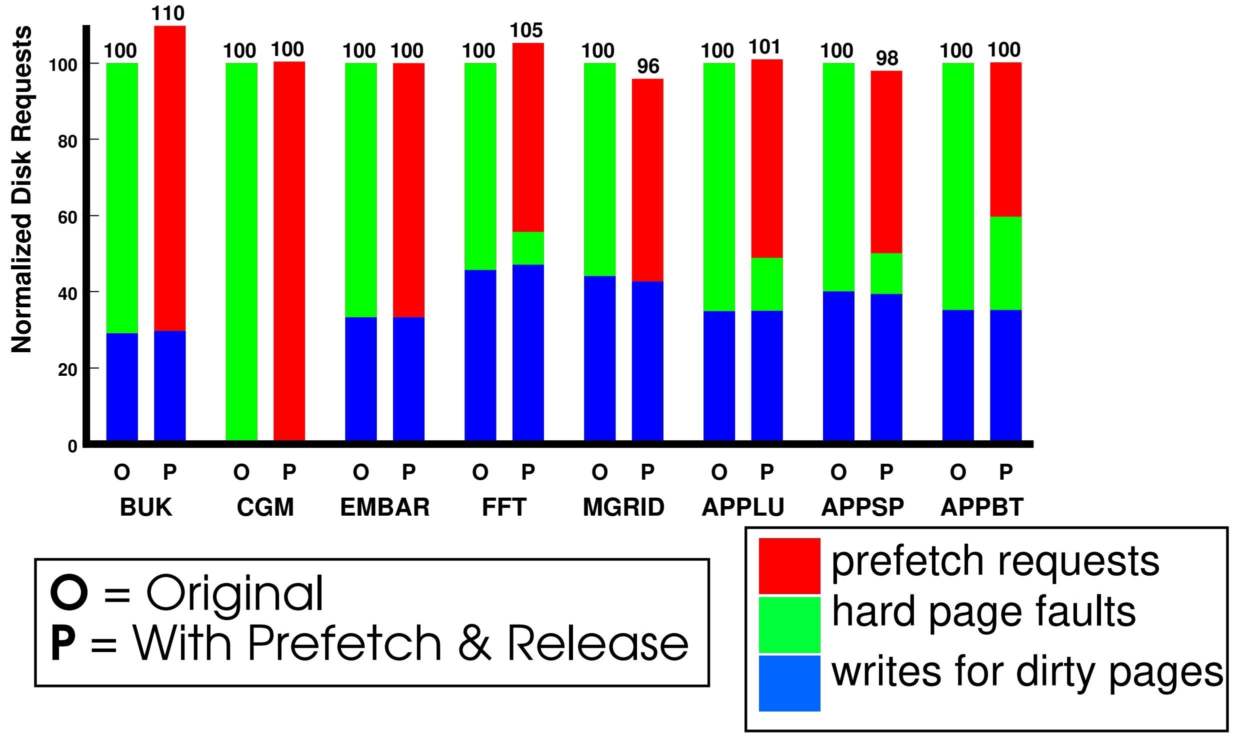
W = Prefetch with run-time layer
WO = Prefetch without run-time layer

- half the benchmarks do worse than without prefetching!

☞ **run-time layer is clearly essential**

Impact on Available Resources

Disk Traffic:

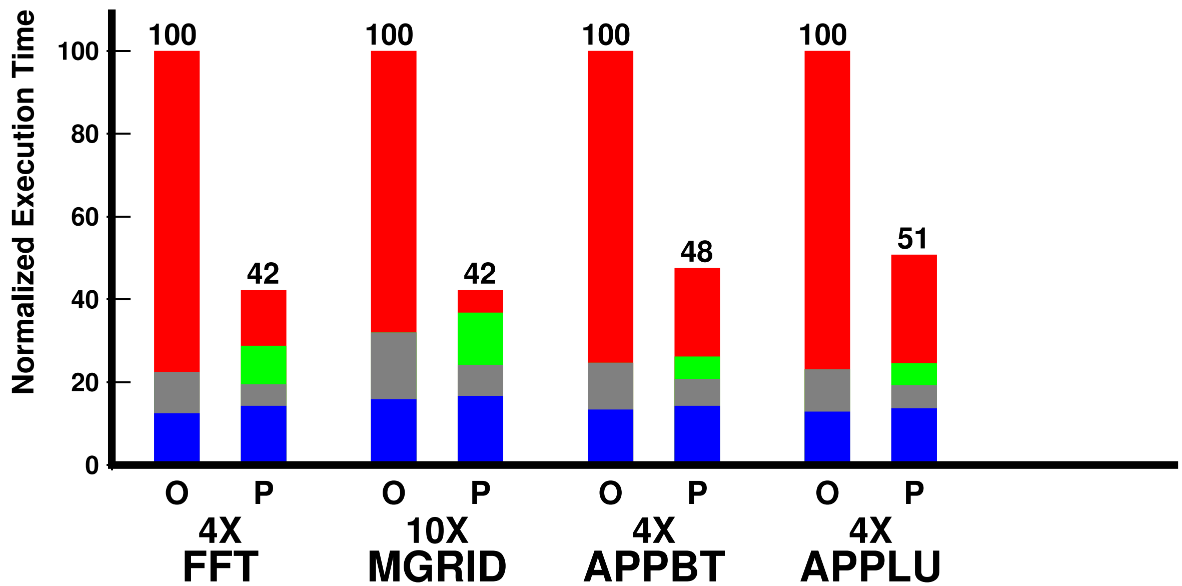


☞ *original page faults replaced by prefetches*

Available Memory:

- average free memory increases due to *release*
  - for EMBAR, 98% with release vs. 22% in original

Larger Problem Sizes



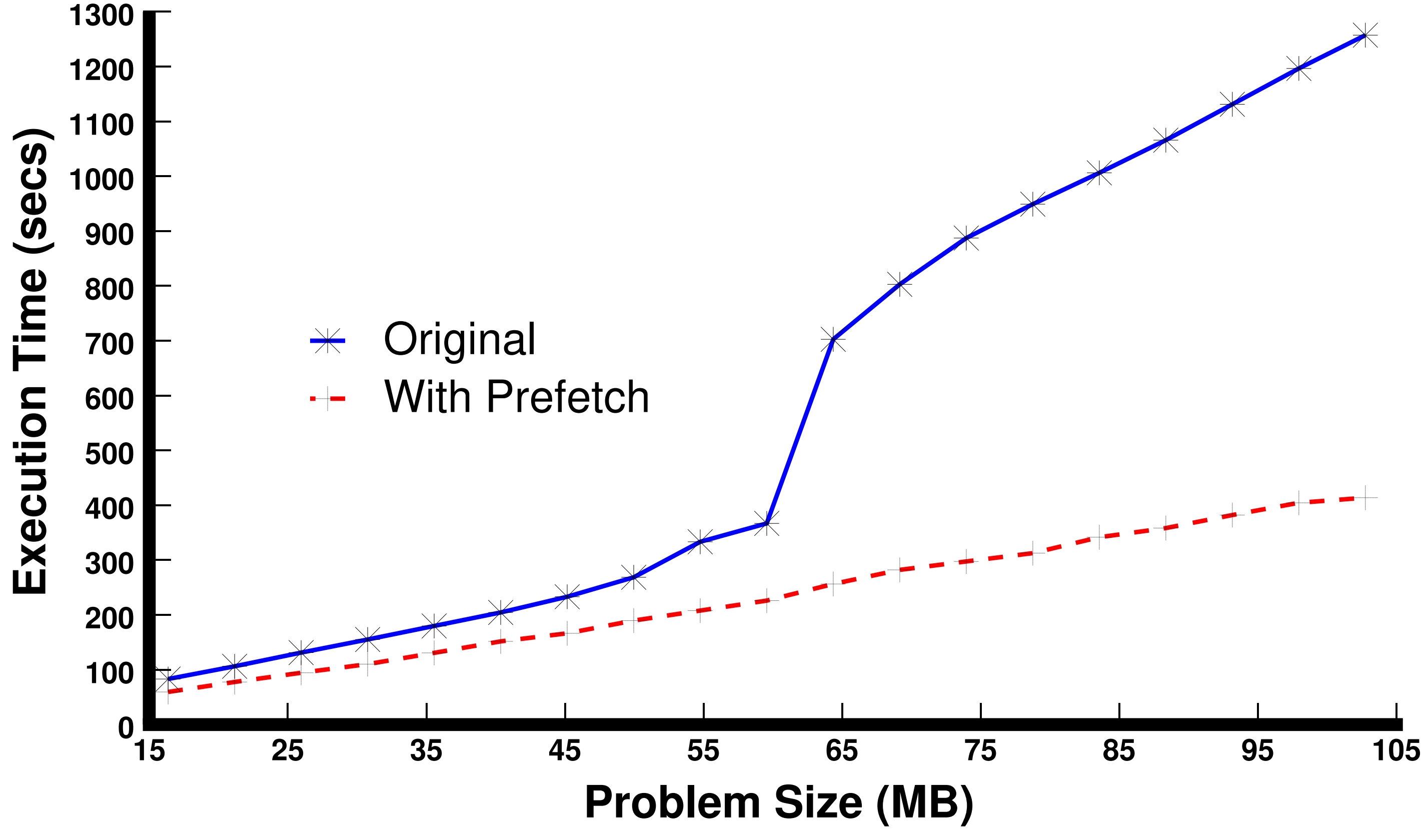
O = Original
P = With Prefetch & Release

I/O stall time
system pf overhead
system time
user time

- greater memory pressure => more I/O latency

☞ **performance improvements remain large**

An Example Revisited



☞ **Effect of I/O on performance is completely hidden**

Conclusions

- **I/O is a serious problem for numeric applications**
  - paged virtual memory offers poor performance
  - difficult to rewrite program to use explicit I/O
- **Automatic I/O prefetching is an attractive solution**
  - programmer can concentrate on solving problem, not I/O
  - compiler can successfully identify and prefetch a wide range of access patterns
  - operating system provides minimal support
  - run-time layer interacts with OS to reduce overheads
- **Future work:**
  - multiprogramming and parallel applications

☞ **Commercial systems should support these techniques**