MICRO-28

IBM AS/400 Division, Rochester, MN

SPAID: Software Prefetching in Pointer- and Call-Intensive Environments

Mikko H. Lipasti, William J. Schmidt,

Robert R. Roediger, and Steven R. Kunkel

IBM Corporation
AS/400 Division
3705 Highway 52 North
Rochester, MN 55901

Speculatively
Prefetching
Anticipated
Interprocedural
Dereferences

Introduction

- The memory bottleneck is beginning to dominate system performance:
    - Conventional hardware-only caching approaches are no longer sufficient
    - CPU to memory speed mismatch is large and getting much worse (currently roughly 10:1, moving towards 100:1 or even worse)
- Software prefetching has been proposed as an alternative:
    - Compiler inserts nonblocking prefetch directives in advance of loads
    - Memory prefetches are overlapped with other useful computation
- Problem: how does compiler decide when & where to place prefetches?

Related Work

- Primarily for loop-intensive, numeric code, which has compile-time predictable memory reference patterns.
- Some work has been done on integer, call-intensive code:
  - Aggressive placement of prefetches overwhelms memory subsystem and creates conflict misses
  - Other work shows only a few key program locations need to be prefetched to avoid large percentage of misses
- We propose a simple heuristic that works well for C++ code, not so well for C code:
  - Heuristic: prefetch pointer parameters at procedure call sites.
- Hypotheses:
  - Pointer parameters tend to get dereferenced, so prefetching is useful
  - Call site is early enough to allow prefetch to complete before dereference occurs

Benchmarks

Bench mark	Description (Language)	Input Set
xlisp	Lisp Interpreter (C)	Six queens lisp program
gcc	Phase cc1 of Gnu C Compiler (C)	insn-recog.c
groff	Text Formatter (C++)	groff man page (40K)
idl	OMG IDL parser (C++)	somcls.idl definition file
gperf	Hash Generator (C++)	scrabdict.200 (200 words)
sched	Instruction scheduler (C++)	eightq C program object
spaid	Cache Model (C/C++)	short trace of gcc (17K)

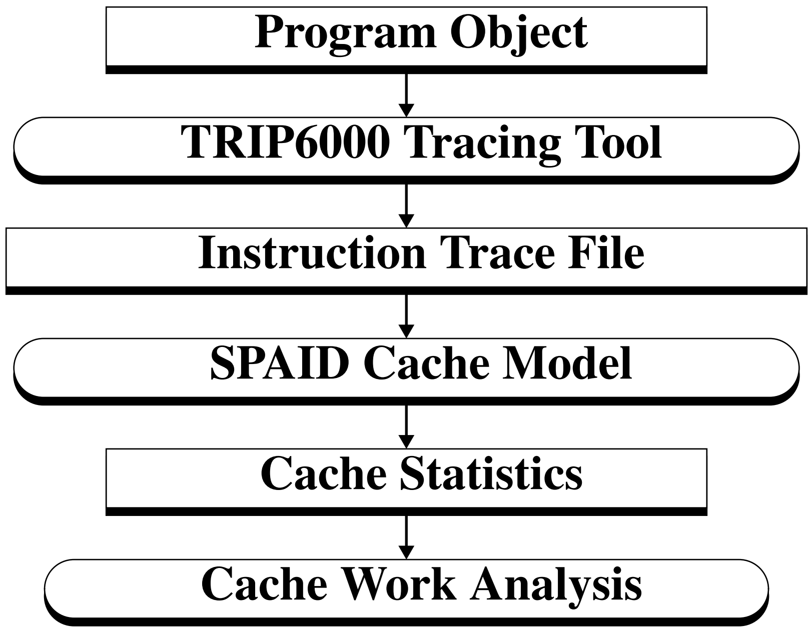
- Call- and pointer-intensive C and C++ programs

Benchmark Characteristics

Bench mark	Run Length	Call Sites	Proc. Length	Touch Sites (% calls)
xlisp	52.10M	1.36M	38.33	1.15M (85%)
gcc	146.14M	1.94M	75.23	1.36M (70%)
groff	118.90M	5.55M	21.42	4.25M (77%)
gperf	7.81M	222K	35.22	105K (47%)
idl	10.84M	356K	30.45	331K (93%)
sched	78.21M	4.38M	17.87	3.61M (83%)
spaid	99.43M	5.03M	19.78	5.01M (100%)
Total	**513.43M**	**18.84M**	**27.26**	**15.82M (84%)**

- Average procedure length: 27 instructions
- Calls that pass one or more pointers: 84%
- All programs run to completion
- Only user-state instructions are traced

Experimental Framework



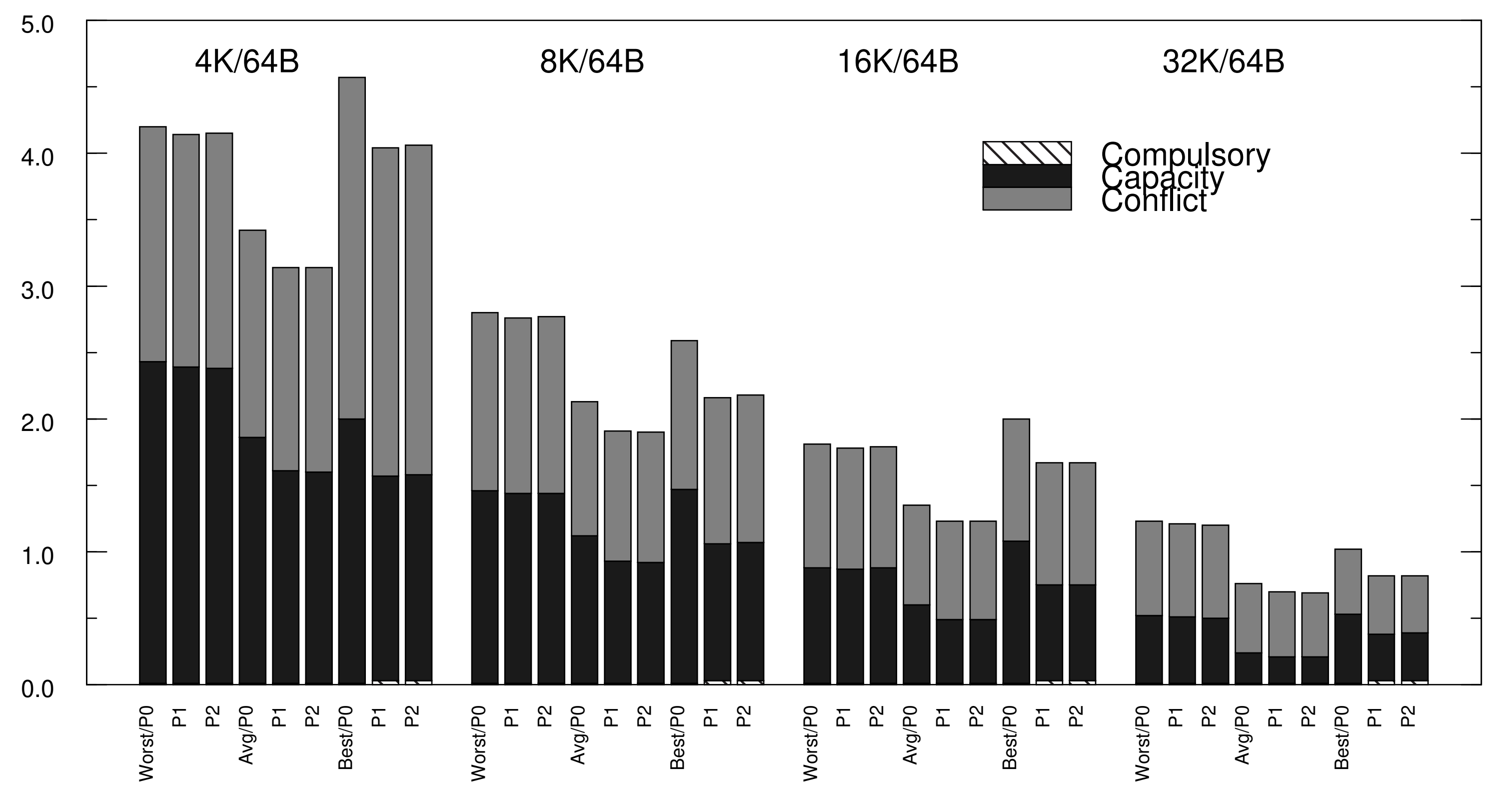
- Traces are modified to include prefetches at call sites
- Cache statistics include prefetch->load distance distribution

Cache Work Analysis

$$CW = I \times CPI_{\infty} + I_m \times I_l + D_m \times D_l + \sum_{i=0}^{D_l} M_i \times (D_l - i)$$

- Cache work approximates the total number of CPU cycles, including CPU stalls waiting for cache misses and incomplete prefetches
- CPI_{∞} is conservatively set to 1
- $\{I,D\}_m$ and $\{I,D\}_l$ are the number of I and D misses (without preceding prefetches) and the I and D miss latencies
- M_i is the number of prefetches that were initiated i cycles ahead of a reference to the cache line being prefetched
- The summation term accounts for incomplete prefetches

Results - Cache Misses



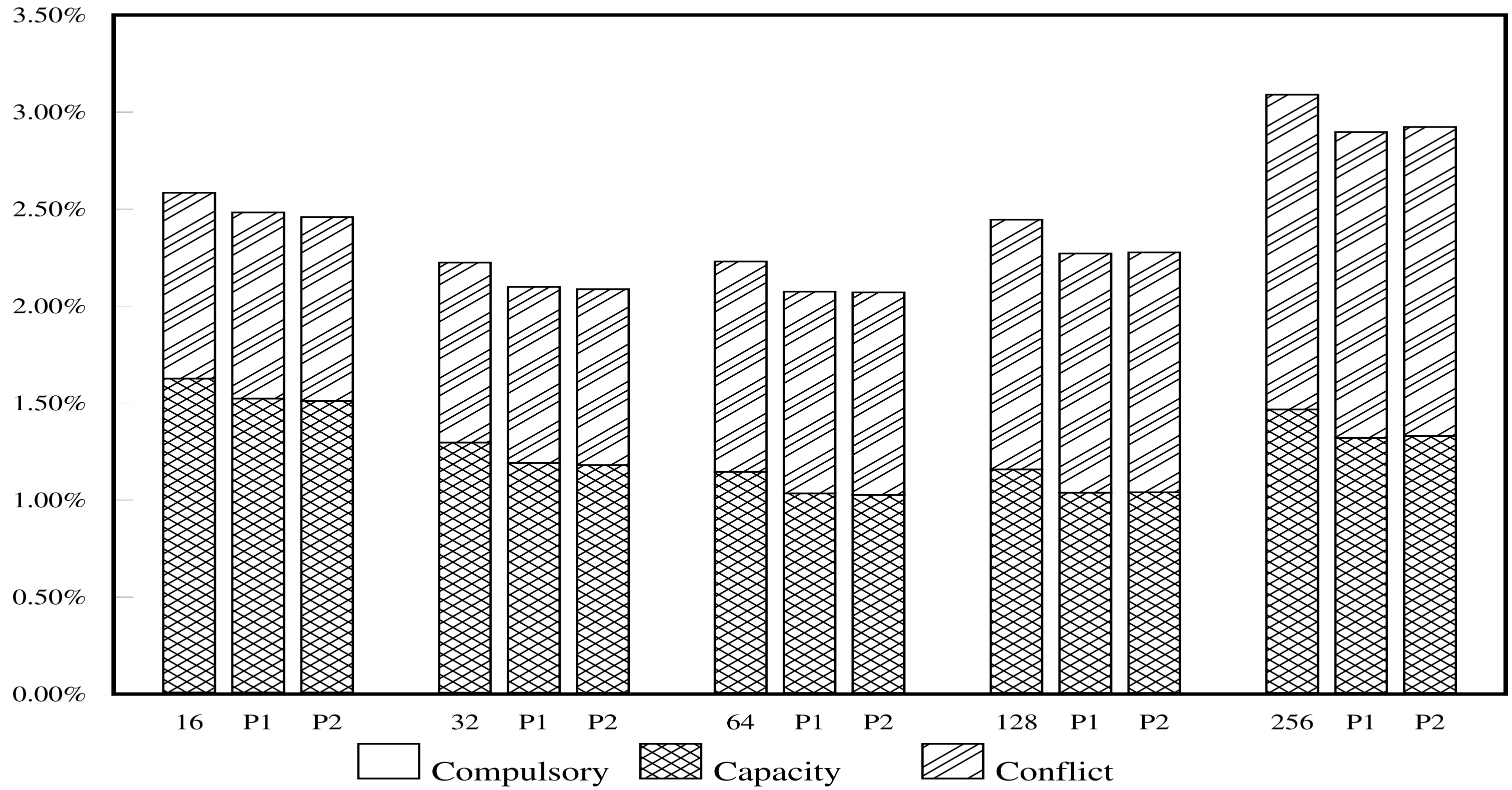
- 5-7% average reduction, 12-20% best case (idl), 1.5% worst case (gcc)
- Allowing a second prefetch per call site buys almost nothing

Results - Miss Rates and % Useful

Bench mark	Prefetch 1 Pointer Per Call Site			Prefetch 2 Pointers Per Call Site		
	Pref. Count	Miss Rate	Use-ful	Pref. Count	Miss Rate	Use-ful
xlisp	1.15M	8.8%	75.3%	1.89M	7.4%	60.0%
gcc	1.36M	9.9%	61.1%	2.27M	5.5%	41.9%
groff	4.25M	10.4%	73.5%	7.00M	8.6%	41.0%
gperf	105K	31.1%	83.1%	162K	22.4%	77.7%
idl	331K	20.5%	52.8%	633K	13.3%	39.0%
sched	3.61M	4.3%	63.9%	4.60M	4.9%	51.6%
spaid	5.01M	7.0%	47.5%	7.73M	5.8%	46.7%
Σ/GM	**15.82M**	**10.8%**	**64.2%**	**24.29M**	**8.4%**	**49.7%**

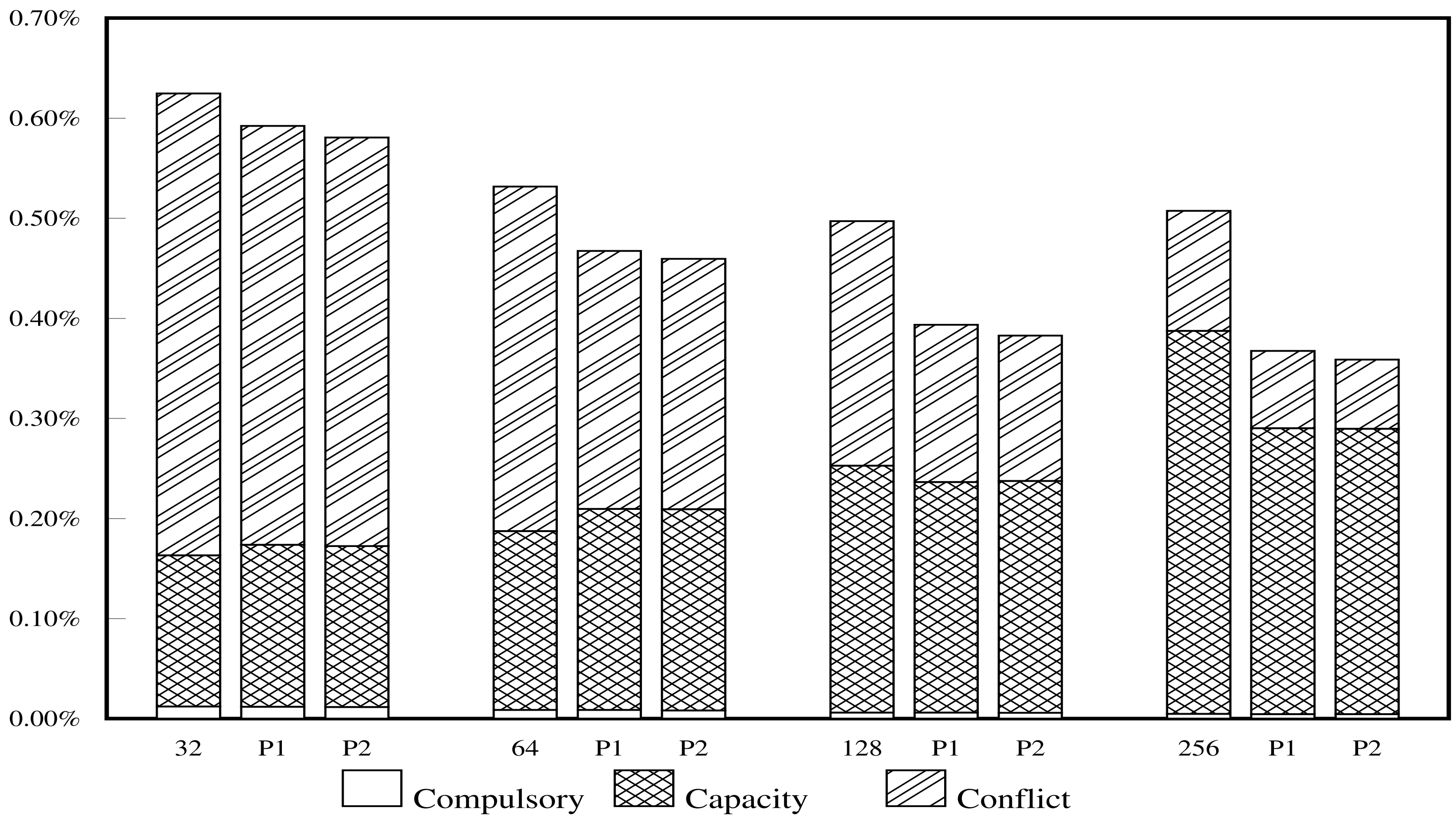
- Prefetching a second pointer is not beneficial:
    - Lower miss rate per prefetch
    - Fewer are useful (prefetched line is actually referenced)

Results - Effects of Line Size



- 64B line is the best choice
- SPAID is robust: effectiveness is 4-7% for all line sizes shown

Results - Effect of Line Size



IBM AS/400 Division, Rochester, MN

- *spaid* benchmark prefers long lines, due to very large objects (directories)

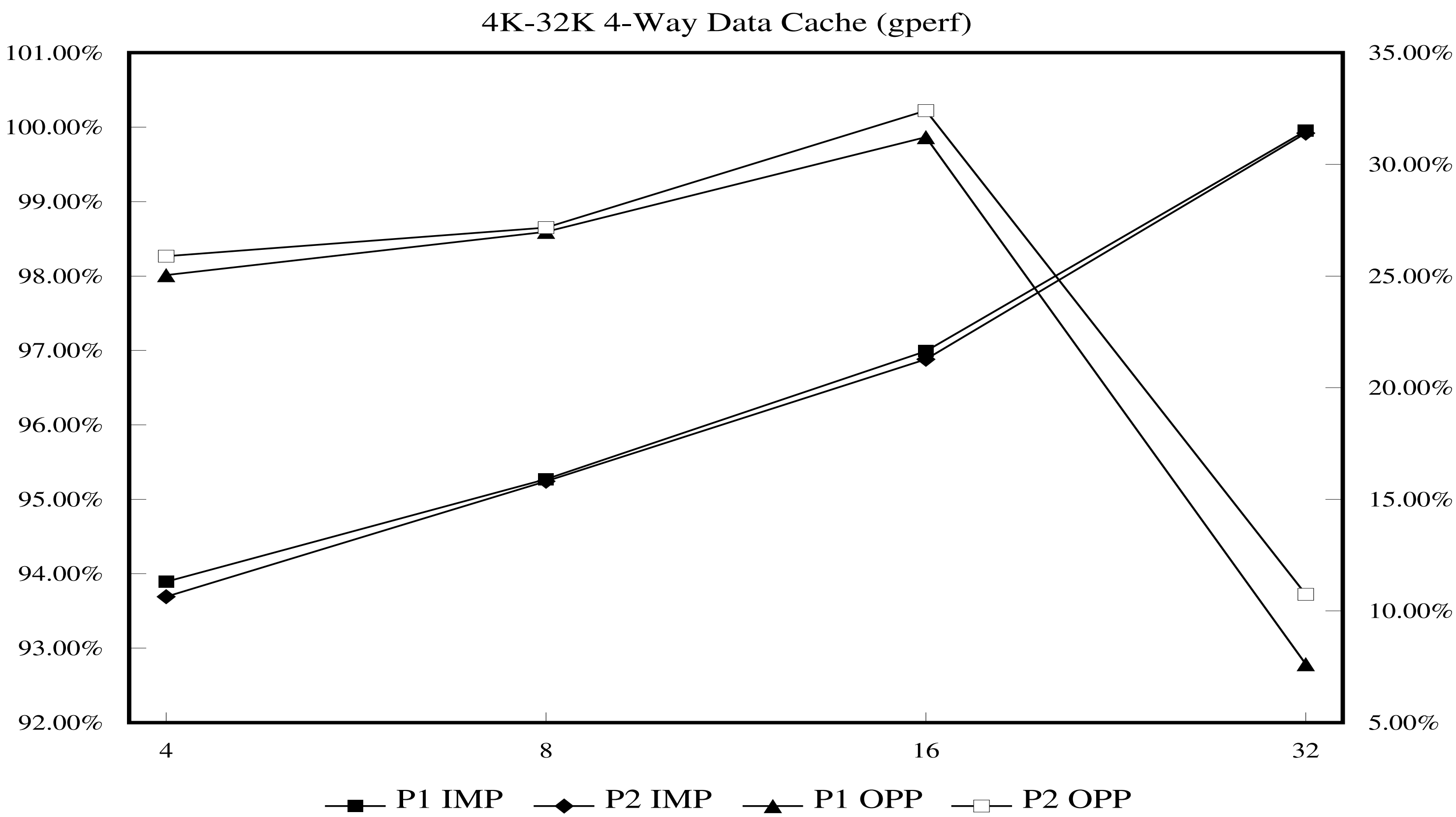
Speculatively
Prefetching
Anticipated
Interprocedural
Dereferences

Results - Cache Work

Bench mark	Base WPI	Prefetch 1 Pointer Per Call Site			Prefetch 2 Pointers Per Call Site		
		WPI	IMP	OPP	WPI	IMP	OPP
xlisp	1.29	1.27	98.5%	18.8%	1.27	98.3%	22.0%
gcc	1.72	1.71	99.9%	0.8%	1.71	99.9%	0.8%
groff	2.12	2.12	99.6%	3.6%	2.11	99.5%	4.8%
gperf	1.28	1.22	95.3%	27.0%	1.22	95.2%	27.2%
idl	1.48	1.43	97.2%	15.7%	1.43	96.8%	18.1%
sched	1.41	1.41	99.7%	5.5%	1.41	99.4%	10.3%
spaid	1.45	1.43	98.6%	9.1%	1.43	98.1%	12.2%
Mean	**1.66**	**1.65**	**98.3%**	**7.3%**	**1.65**	**98.1%**	**9.1%**

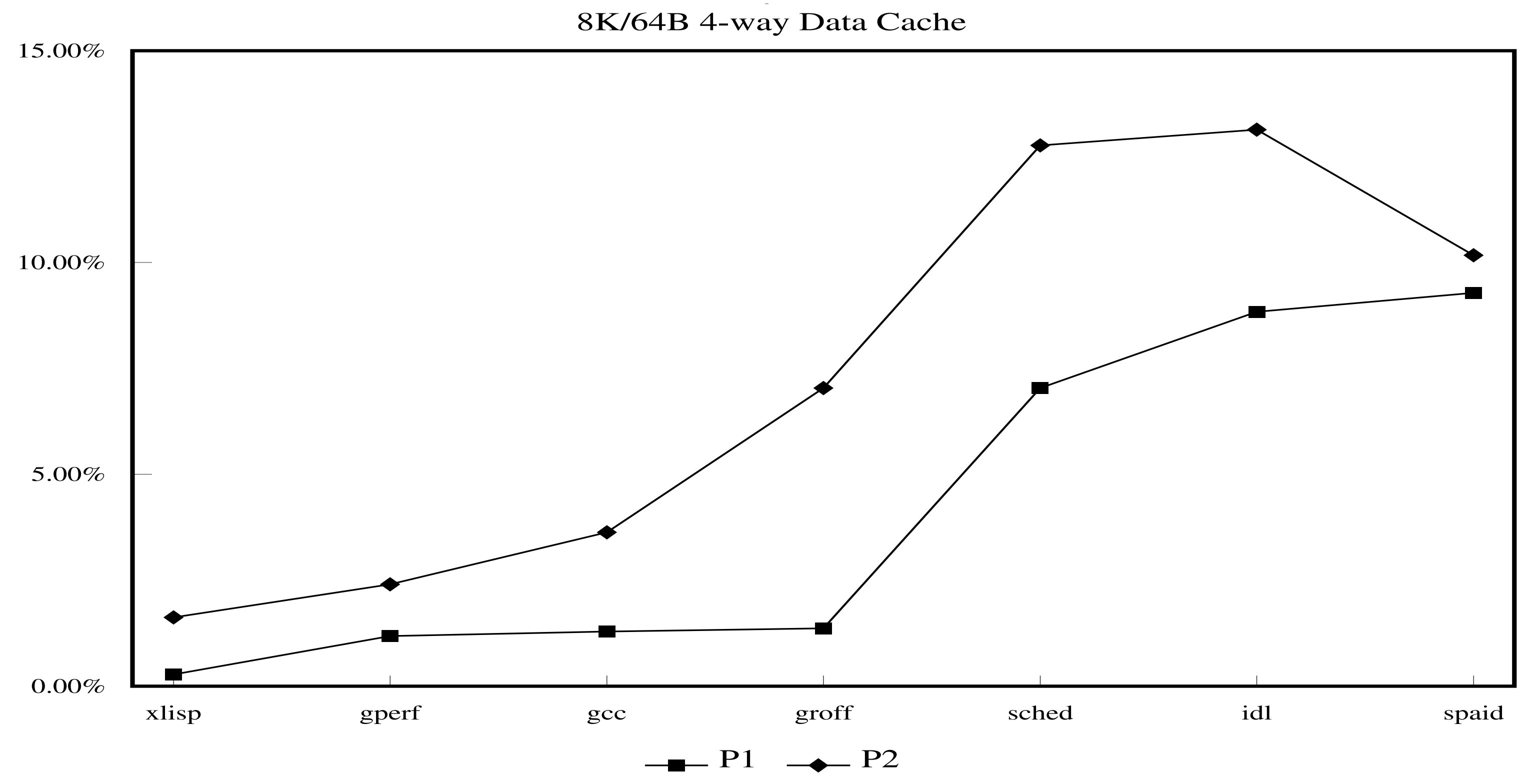
- Best case, 27% of stall cycles eliminated, 7-9% average case
- Overall improvement less than 2%, but *OPP* is the meaningful measure

Results - Cache Work vs. Cache Size



- IMP and OPP deteriorate gracefully until 32K, which has 0.1% miss rate

Results - Memory Traffic



- Worst case increases less than 10% for single prefetch
- Multiple prefetches in many cases bring in unused lines

Conclusions and future work

- Results are encouraging:
  - Simple, easy-to-implement heuristic
  - Can remove significant fraction of data cache miss penalties
  - No dramatic increase in memory bandwidth
- Heuristics for choosing which pointers to prefetch:
  - For C++, choosing "this" pointer works well
  - For C and other languages, the problem is more difficult
  - Choosing more than one per call site is even more problematic
- Further validation needed:
  - Collect data on actual system and actual compiler implementation
  - Implement cycle-by-cycle simulation model instead of statistical one