PHIPAC: Portable HIgh Performance Ansi C

Krste Asanovic, Jeff Bilmes,

James Demmel, Dominic Lam,

Nelson Morgan

Department of EECS, U.C. Berkeley
and
International Computer Science Institute

bilmes@cs.berkeley.edu, krste@cs.berkeley.edu

Outline

- Motivation and Background
- Methodology
- Performance Results

Motivation and Background

- Cheap fast matrix-vector C-based library for various workstations.
- Efficient workstation-based T0-compatible library.
- Keep Krste from finishing a course project.
- Make BERP run faster.
- Get to talk at the Castle retreat.

Goals

- BLAS compatible interface.
- A fast, portable, public-domain, tunable matrix-vector library in C.

An Observation

- Programmers often write C code that overspecifies the order in which operations must occur.
- C compilers, constrained by C semantics, must obey these overspecifications. This makes it hard or impossible to fully optimize.

Methodology

- Make the compiler's job easy: only rely on compiler for instruction selection, scheduling, and register allocation.
- Generate C code that is easy to optimize: remove false dependencies, use machine sympathetic constructs.
- Write parameterizeable C-code generators, then tune to achieve optimal performance.

Target Machine Model:
Generic RISC Workstation

- Many FP registers (16-64)
- Multi-instruction FPU & memory latency (pipelined/superscalar)
- Roughly balanced at 1 mul + 1 add + 1-2 load/store
- Base + offset addressing mode
- Cached (at least L1, usually L2, perhaps L3)
- Multi-word cache lines
- Integer multiply/divide slow relative to addition
- Branches are costly

Target Machine Model (cont.)
Modern C compiler

- Assume the C compiler is very good at:
  - instruction selection
  - instruction scheduling
  - register allocation

PHIPAC Tricks

Exploit Many FP Registers

- Name regs to remove false dependencies

```
a[i] = b[i]+c;                  f1 = b[i]; f2 = b[i+1];
a[i+1] = b[i+1]+c;              a[i] = f1+c; a[i+1] = f2+c;
```

- Reduce Memory Bandwidth requirements

```
float f0,f1,f2;  /* locals as registers */
/* explicitly move values from mem to regs */
f0 = filt[0]; f1 = filt[1]; f2 = filt[2];
while ( ... ) {
   *res++ = f0*sig[0] + f1*sig[1] + f2*sig[2];
   sig++;
}
```

PHIPAC Tricks

Multi-Instruction FP Latency
(pipelined and/or superscalar)

- Expose independent ops to allow good instruction scheduling.

```
f1 = f2*f3;
f4 = f5*f6;
f7 = f8*f9;
```

- Manual loop unrolling to expose ILP

PHIPAC Tricks

Roughly balanced at 1 mul + 1 add + 1-2 load/stores

- Aim for this mix whenever possible

```
f1 = f2*f3;
f4 = f5+f6;
f7 = *ptr;
```

PHIPAC Tricks

Assume intrinsic base+offset addressing modes are available.

- Avoid pointer updates with constant stride.

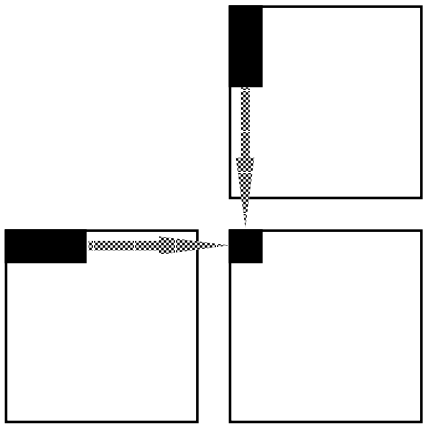
```
f0 = *r8++;
f1 = *r8++;
f2 = *r8++;
```

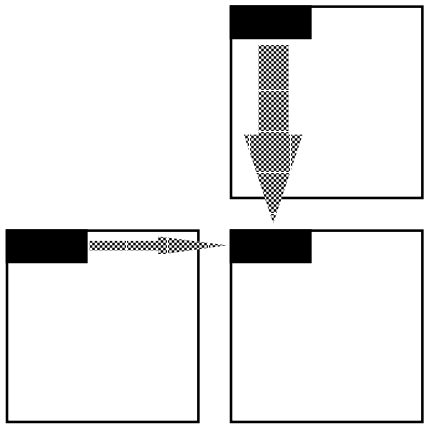
```
f0 = *r8[0];
f1 = *r8[1];
f2 = *r8[2];
```

PHIPAC Tricks

Multi-word cache lines

- Exploit spatial locality via restructuring for unit stride.





PHIPAC Tricks

Integer multiply/divide slow relative to addition.

- Use pointer updates, not expensive subscript expression.

```
for (i = ... ) {
   row_ptr = &p[i*col_stride];
   ...
}
```

```
for (i= ...) {
   ...
   row_ptr += col_stride;
}
```

PHIPAC Tricks

Branches are Costly

- Use manual loop unrolling to amortize cost.
- Use cheaper == or != loop termination tests.

```
for (a = start_ptr; a != end_ptr; a++)
   ...
```

Generators

- Generator: program that outputs C code for a routine given a set of parameters.
  - mm_gen -cb 2 6 10 -cb 24 10 4
- Find optimal performance
  - What is fastest routine instance for a particular compiler+machine combination? Search scripts find optimal parameter settings.
- Portability
  - How well does any one routine do across different compiler+machine combinations?

Results

- matrix-matrix multiply on
  - IBM RS6000/590
  - Sparc 20/61
  - SGI Indigo R4000
- dot-product on
  - Sparc 20/61
  - SGI Indigo R4000

Status

- Generators so far:
  - matrix-matrix multiply (+ all transpose variants)
  - matrix-vector multiply
  - vector-matrix multiply
  - dot product
  - saxpy
  - convolution (FIR filter)
  - outer-product
- mm search scripts found optimal parameters on IBM RS6k, Sparc20/61, Sparc-2, Sparc-2+Weitek, SGI Indigo R4k

Summary

- Can write fast matrix-vector code in portable ANSI C.
- Can write fast code for many machines via generators.
- Consequences for library writers and compiler writers using C as a target language.

Future Work

- Integration into Bo Kagstrom's DGEMM based BLAS.
- Find the "best" portable code for each routine.
- Run search scripts on more machines.
- Generators for other routines: FFT, IIR filter, sorts, etc.