Getting More From Your Multicore

Michael S. Noble
M.I.T. Kavli Institute For Astrophysics
March 1, 2007
(updated March 14 & March 23)

Underutilization

Many of our machines have multiple CPUs

Used for occasional extra ISIS process ...

... or compiling XSPEC while editing text ...

... et cetera

Are They Being Used To Best Advantage?

System-Level Parallelism

System software manages & farms out ...

Whole programs

To multiple CPUs or machines

Examples: Linux OS, ISIS with PVM

Easy for Users
Existing Codes Not Extensively Rewritten
Apps "Don't Know" About Parallelism

Program-Level Parallelism

Program explicitly manages & farms out ...

Parts of itself

To multiple CPUs

Examples: POSIX threads, FLASH with MPI

Usually MUCH harder
Existing Codes Usually Need Extensive Changes

"Why Threads Bad Idea" (Ousterhout, 1995)

MultiCore Ubiquity

The difficulty of making PARALLEL versions

of SEQUENTIAL codes

has long been a bane of HPC

Parallelism hardly used in common astro S/W

EG: XSpec models run sequential on my dual-CPU desktop

How will we use emerging multicore systems?

OpenMP

Observation: most time spent in loops

```
#pragma omp parallel for

for (i=0;  i <N; i++)
    result[i] = compute(...);
```

Why not parallelize with C *#pragma* ?

(more directives exist to parallelize other constructs)

OpenMP

Easy To Understand & Apply

Same code for sequential & parallel

#pragma ignored
when unsupported by compiler

Supports BIG 3 Languages: C, C++, FORTRAN

ADVANTAGES

OpenMP

Special compilers

Historically Commercial / Expensive

Virtually no open-source support

Unclear how to exploit in scripting context

DISADVANTAGES

OpenMP

Support added to GCC 4.2

libgomp

Still in unstable/testing

Public release in 2007?

HOWEVER ...

SLIRP

Already generates vectorized wrappers

Observation: vectorization is just loops

```
void slirp_vectorized_cos(void)
{
   ...
   for (i=0; i < num_iters; i++)
        vec_retval[i] = cos(arg1[i]);
   ...
   return vec_retval;
}
```

Trivial to add OpenMP to SLIRP

Easy way to exploit multicore!

SLIRP

linux% slirp -openmp math.h

```
#pragma omp parallel for
for (i=0; i < num_iters; i++)
    vec_retval[i] = cos(arg1[i]);
```

Adds one line to each vectorized wrapper

Linux GCC 4.2 Performance

Native

```
svoboda% isis
isis>  x = [-PI*100000: PI*100000: .05]

isis>  tic;   s = sin(x);   toc
0.96872
```

SLIRP
-openmp

```
isis>  import("pmath")

isis>  tic;   s = sin(x);   toc;
0.55599
```

Nearly 2X faster on 2 CPUs (87% efficient)

Solaris 9 Performance

Native

```
tdc@cfa%  setenv OMP_NUM_THREADS 4

tdc@cfa%  slsh
slsh>  x = [-PI*100000: PI*100000: .05]

slsh>  tic;   s = cos(x);   toc
6.7933
```

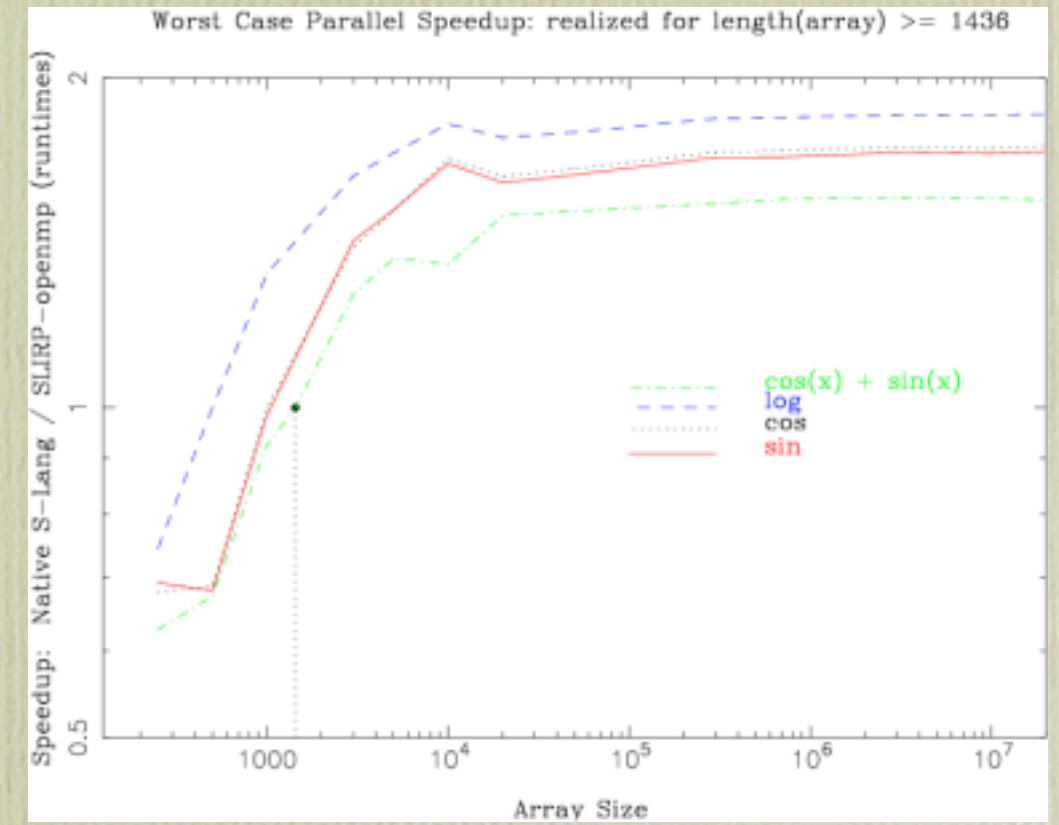
SLIRP -openmp

```
slsh>  import("pmath")

slsh>  tic;   s = cos(x);   toc;
1.7823
```

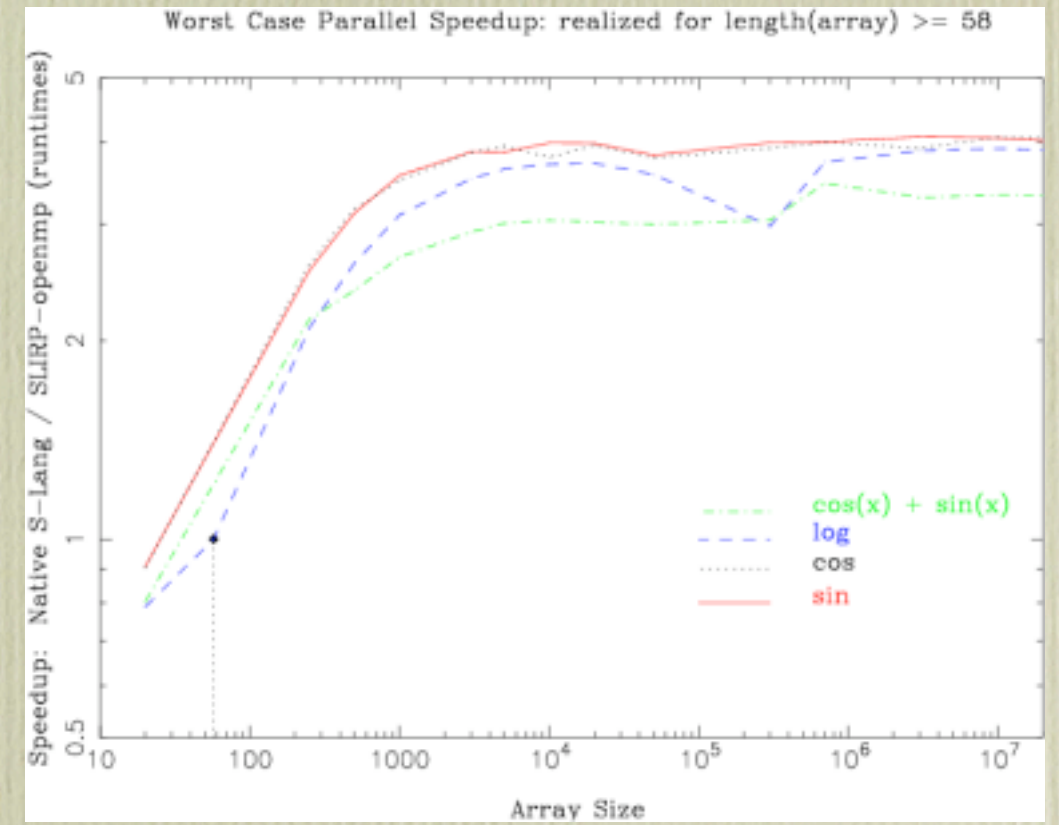
Nearly 4X faster on 4 CPUs (95% efficient)

Performance Inflection Point



Dual 1.8 Ghz Athlon CPUs
GCC 4.2 Debian Linux

Performance Inflection Point



4 (of 8) 750Mhz sparcv9 CPUs
Sun Studio 9 Solaris 5.9

Applicability

Transparent: parallel functions have same name
You don't have rewrite your codes

Growing a module of vector-parallel functions
(including non-numeric, like `strlen()`, `atoi()`, et cetera)

Should be very useful on our 52-core Hydra cluster
(coming soon!)

Current Limitations

SLIRP -openmp does not work for FORTRAN functions (yet)

Vectorizing arrays must be sufficiently large to gain performance (see docs for plots of speedup inflection points for 2 & 4 CPUs)

You have to build prerelease version of GCC 4.2 or 4.3

Some platform compilers (e.g. Solaris cc) require parent app be linked with OpenMP to use OpenMP-aware module.

Type casting (e.g. float to double) can reduce performance gains

S-Lang `Complex_Type` is not supported (yet)