

PAPI: Performance Application Programming Interface

Walt Mankowski (waltman@pobox.com)

2 May 2007

Analyzing performance

Linux comes with a lot of ways to tell how long a program is running:

- `time(1)`
- `time(2)`
- `clock(2)`
- `times(2)`
- `gprof(1)`

Analyzing performance

- These are all relatively high-level tools
- They tell what's slow, and how long it's taking, but they don't tell you **why** it's slow.

Modern CPUs can track a lot more low-level data

- cycle count
- instruction count
- floating point instruction count
- pipeline stalls
- L1 cache hits/misses
- L2 cache hits/misses
- TLB misses
- hardware interrupts

PAPI

- Runs on a number of hardware platforms and operating systems
- Provides consistent high-level interface (C and Fortran) to CPU performance data
- Requires Perfctr kernel patch on Linux

Using PAPI

- Install Perfctr kernel patch
- Install PAPI
- Add PAPI functions to your application
- Link to PAPI library with “ -lpapi ”

PAPI API

PAPI has over 40 functions in its API, but you only really need a few:

- PAPI_library_init()
- PAPI_num_counters()
- PAPI_query_event()
- PAPI_start_counters()
- PAPI_read_counters()
- PAPI_flops()

PAPI_library_init()

- Initialize the PAPI library.

```
if (PAPI_VER_CURRENT !=  
    PAPI_library_init(PAPI_VER_CURRENT))  
    ehandler("PAPI_library_init error.");
```

PAPI_num_counters()

- Check how many counters this CPU can monitor

```
const size_t EVENT_MAX = PAPI_num_counters();
```

PAPI_query_event()

- Check if the CPU can monitor the event you're interested in

```
if (PAPI_OK != PAPI_query_event(PAPI_TOT_INS))
    ehandler("Cannot count PAPI_TOT_INS.");

if (PAPI_OK != PAPI_query_event(PAPI_L1_DCM))
    ehandler("Cannot count PAPI_L1_DCM.");

if (PAPI_OK != PAPI_query_event(PAPI_L2_DCM))
    ehandler("Cannot count PAPI_L2_DCM.");
```

PAPI_start_counters()

```
size_t EVENT_COUNT = 3;  
int events[] = { PAPI_TOT_INS, PAPI_L1_DCM, PAPI_L2_DCM };  
PAPI_start_counters(events, EVENT_COUNT);
```

PAPI_read_counters()

```
long long values[EVENT_COUNT];
```

```
if (PAPI_OK != PAPI_read_counters(values, EVENT_COUNT))  
    ehandler("Problem reading counters 1.");
```

```
C = matrix_prod(n, n, n, n, A, B);
```

```
if (PAPI_OK != PAPI_read_counters(values, EVENT_COUNT))  
    ehandler("Problem reading counters 2.");
```

```
printf("%d %lld %lld %lld\n", n, values[0], values[1], values[2]);
```

PAPI_flops()

```
float rtime;  
float ptime;  
long long flpops;  
float mflops;
```

```
if (PAPI_OK != PAPI_flops(&rtime, &ptime, &flpops, &mflops))  
    ehandler("Problem reading flops 1");
```

```
C = matrix_prod(n, n, n, n, A, B);
```

```
if (PAPI_OK != PAPI_flops(&rtime, &ptime, &flpops, &mflops))  
    ehandler("Problem reading flops 2");
```

```
printf("%d %lld %f\n", n, flpops, mflops);
```

How am I doing on
time?

Bonus slides!

Example: Benchmarking Matrix Multiplication

1. Matrix multiplication code from *Numerical Recipes in C*
 - compiled with -O1, -O3, and -O3 -funroll-loops
2. ATLAS (ringer)
 - full optimizations

Numerical Recipes code

```
float **matrix_prod(m1,n1,m2,n2,A,B)
float **A,**B;
int m1,n1,m2,n2;
/*
 * Matrix product.  A is a m1 X n1 matrix with range [1..m1][1..n1]
 * and B is a m2 X n2 matrix with range [1..m2][1..n2].  n1 = m2.
 * C = A * B.
 */
{
int i,j,k;
float **C;

C = zero_matrix(1,m1,1,n2);
for (i=1;i<=m1;i++)
    for (j=1;j<=n2;j++)
        for (k=1;k<=n1;k++)
            C[i][j] = C[i][j] + A[i][k] * B[k][j];
return C;
}
```

Numerical Recipes code

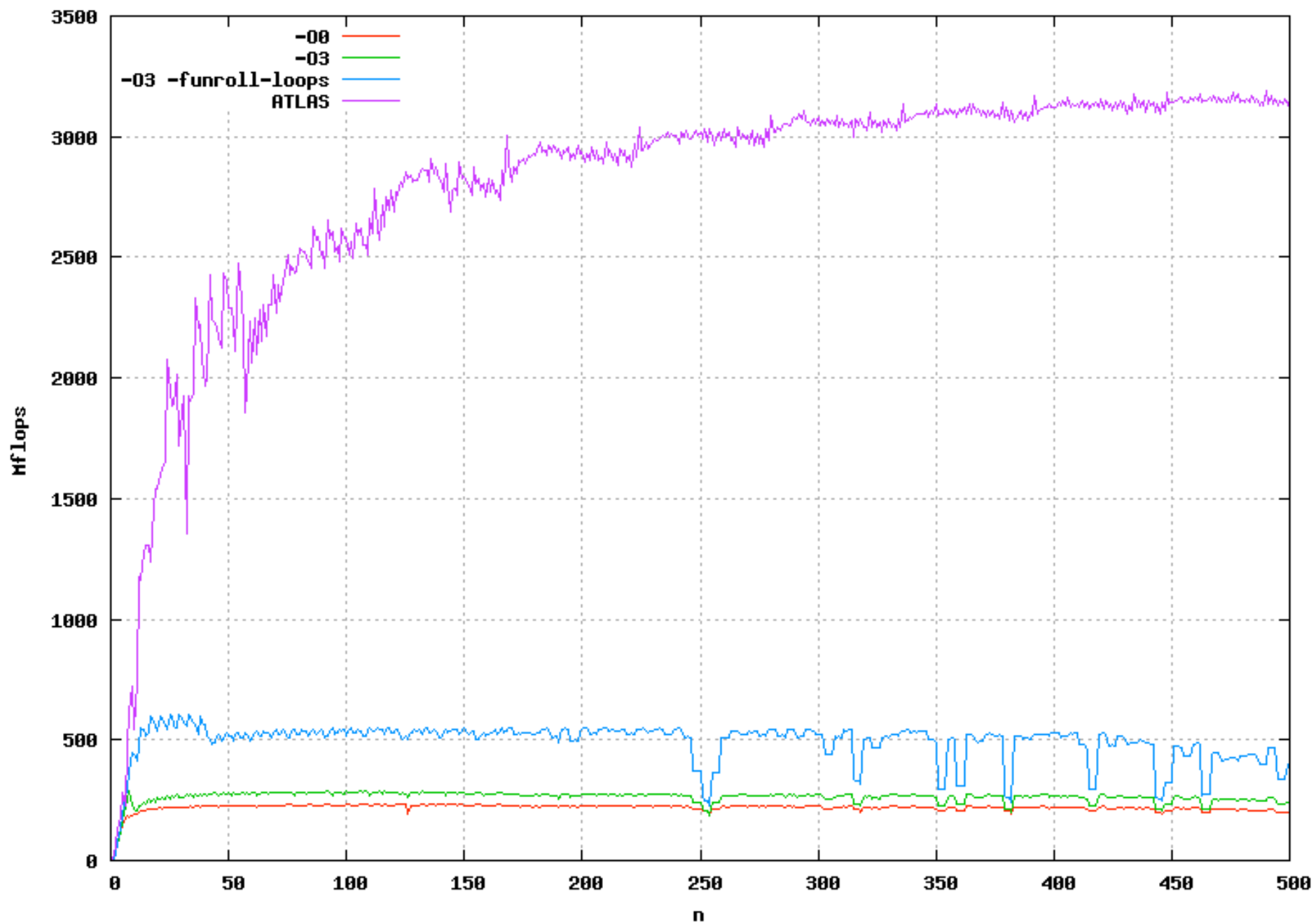
```
float **matrix_prod(m1,n1,m2,n2,A,B)
float **A,**B;
int m1,n1,m2,n2;
/*
 * Matrix product.  A is a m1 X n1 matrix with range [1..m1][1..n1]
 * and B is a m2 X n2 matrix with range [1..m2][1..n2].  n1 = m2.
 * C = A * B.
 */
{
int i,j,k;
float **C;

C = zero_matrix(1,m1,1,n2);
for (i=1;i<=m1;i++)
    for (j=1;j<=n2;j++)
        for (k=1;k<=n1;k++)
            C[i][j] = C[i][j] + A[i][k] * B[k][j];
return C;
}
```

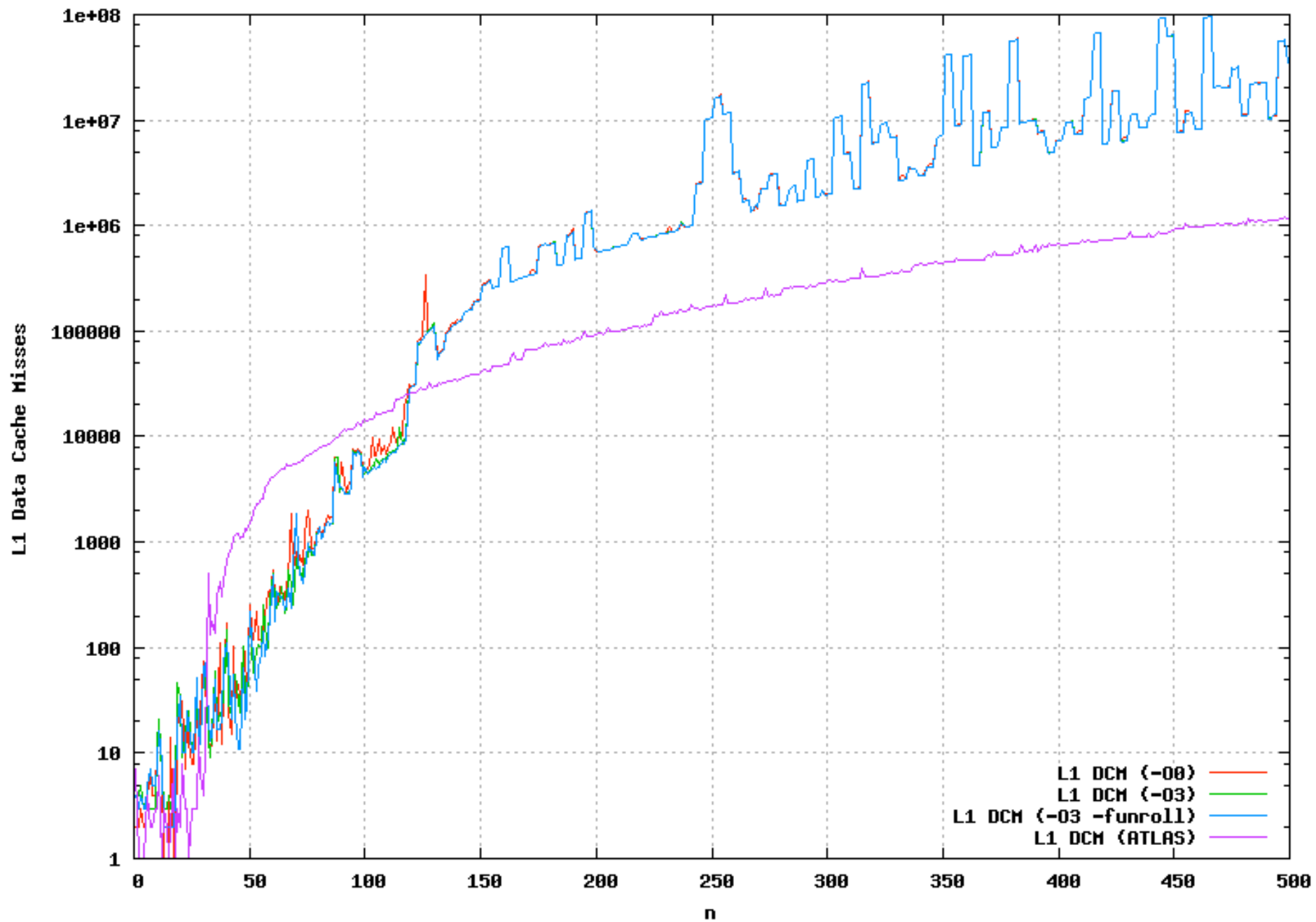
Test Platform

CPU	1.8 Ghz Opteron
RAM	2 GB
Kernel	2.6.13
Distribution	Gentoo
L1 Cache	64 KB
L2 Cache	768 KB
Integer instruction pipeline	12
FP instruction Pipeline	17

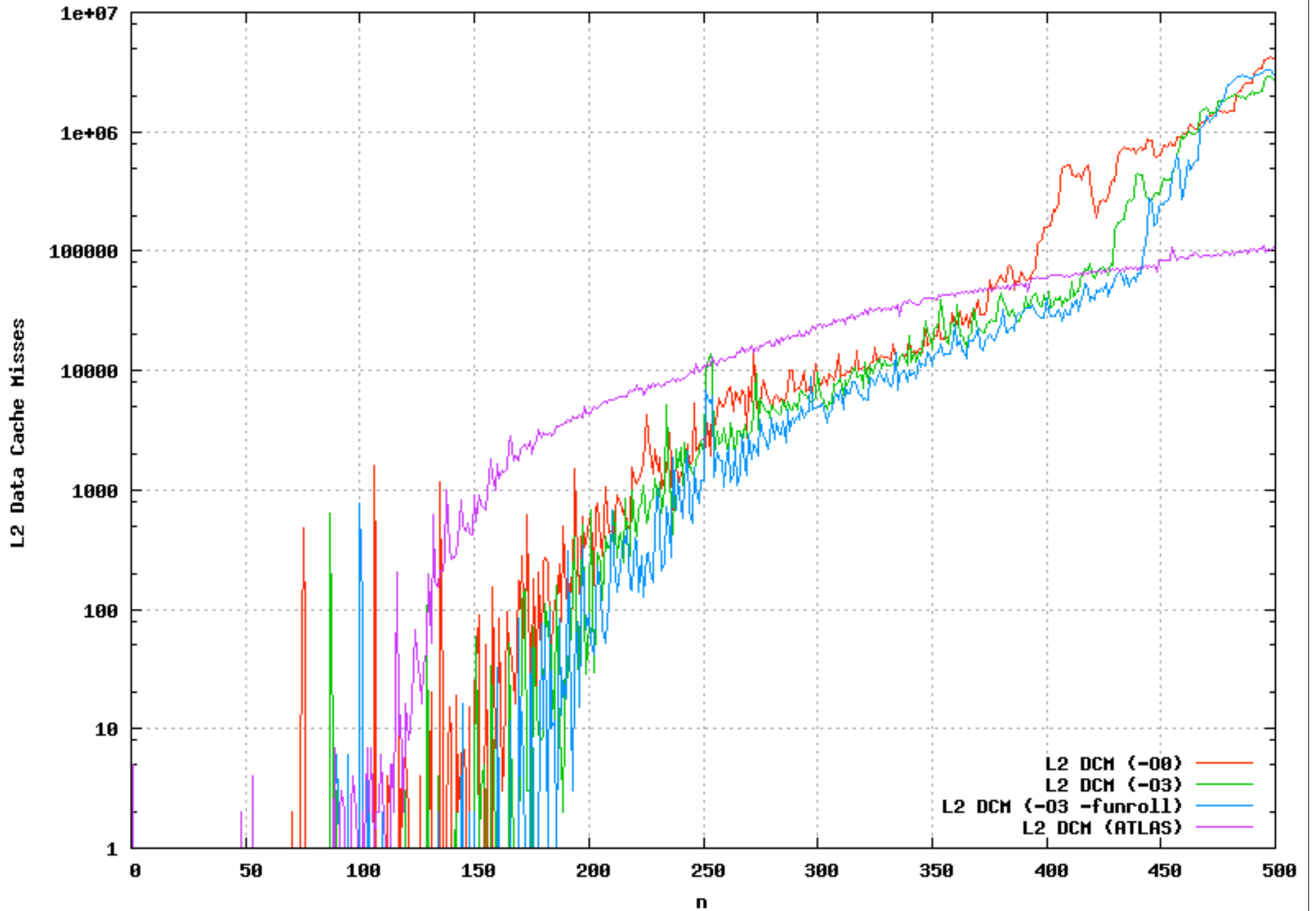
Numerical Recipes Matrix Multiplication
Mflops vs n



Numerical Recipes Matrix Multiplication L1 Data Cache Misses vs n



Numerical Recipes Matrix Multiplication L2 Data Cache Misses vs n



More information

PAPI homepage

- <http://icl.cs.utk.edu/papi/>

ATLAS

- <http://acts.nersc.gov/atlas/>