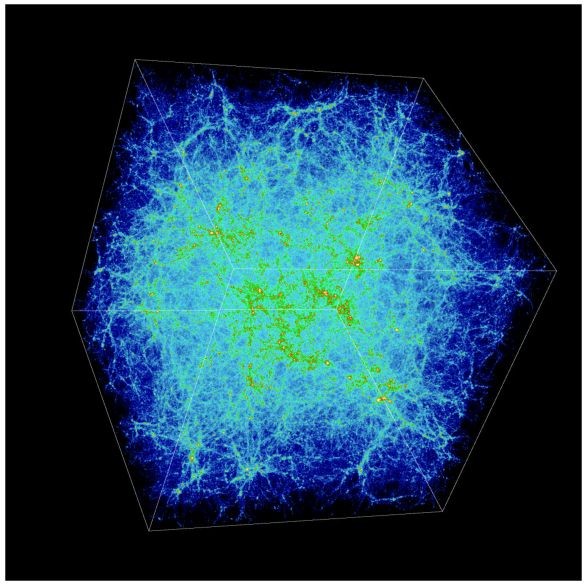
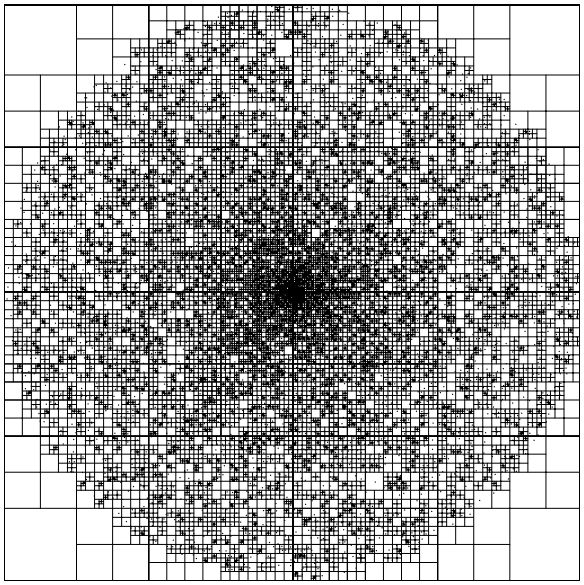
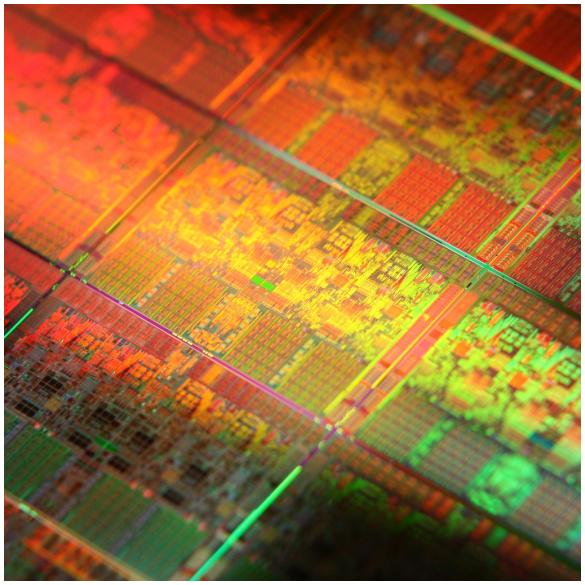
Optimizing the Inner Loop of the Gravitational Force Interaction on Modern Processors

Michael S. Warren
Los Alamos National Laboratory
msw@lanl.gov







Gravitational Force

$$\vec{F}_i = GM_i \sum_{0 \le j \neq i < N} M_j \frac{\vec{r}_i - \vec{r}_j}{\left(||\vec{r}_i - \vec{r}_j||^2 + \epsilon^2 \right)^{\frac{3}{2}}}$$

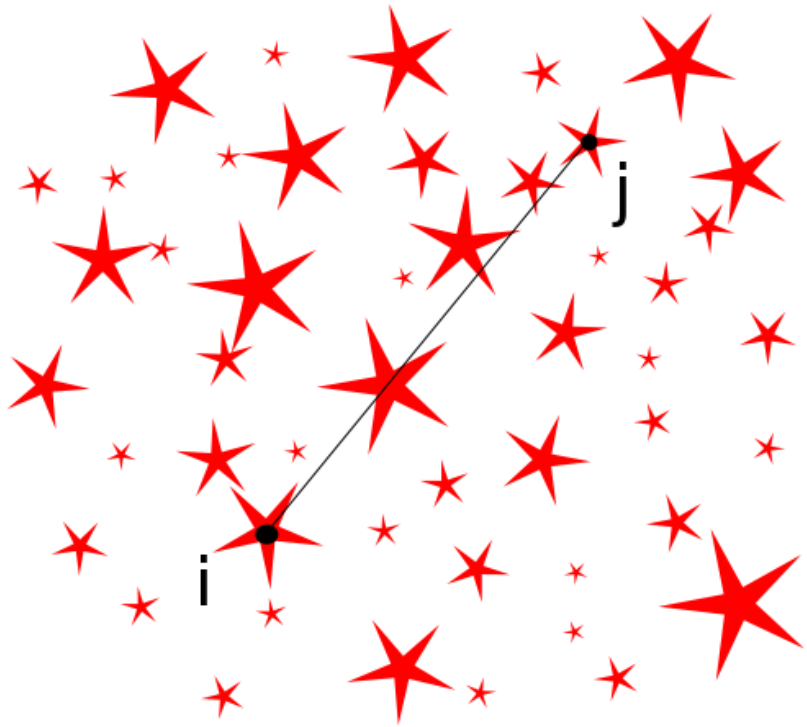


Figure 1: The Gravitational N-body Problem

Inner Loop Performance

Processor	libm	Karp
40-MHz Intel i860		35.2
167-MHz Cray Y-MP	(est) 50.0	
500-MHz Intel P3	87.6	185.6
533-MHz Alpha EV56	76.2	242.2
933-MHz Transmeta TM5800	189.5	373.2
375-MHz IBM Power3	298.5	514.4
1133-MHz Intel P3	292.2	594.9
1200-MHz AMD Athlon MP	350.7	614.0
1800-MHz AMD Athlon XP	609.9	951.9
1250-MHz Alpha 21264C	935.2	1141.0
2530-MHz Intel P4 (icc)	1170.0	1357.0
2530-MHz Intel P4 (SSE)	6514.0	
2600-MHz AMD Opteron (SSE3)	7380.0	
2600-MHz AMD Opteron 8435	13878.0	
2660-MHz Intel Xeon E5430	16343.0	
PowerXCell 8i (single SPE)	16356.0	

Table 1: Mflop/s obtained on our gravitational micro-kernel benchmark.

Should we Optimize?

"We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil."

—Donald Knuth

What is Most Important to Optimize?

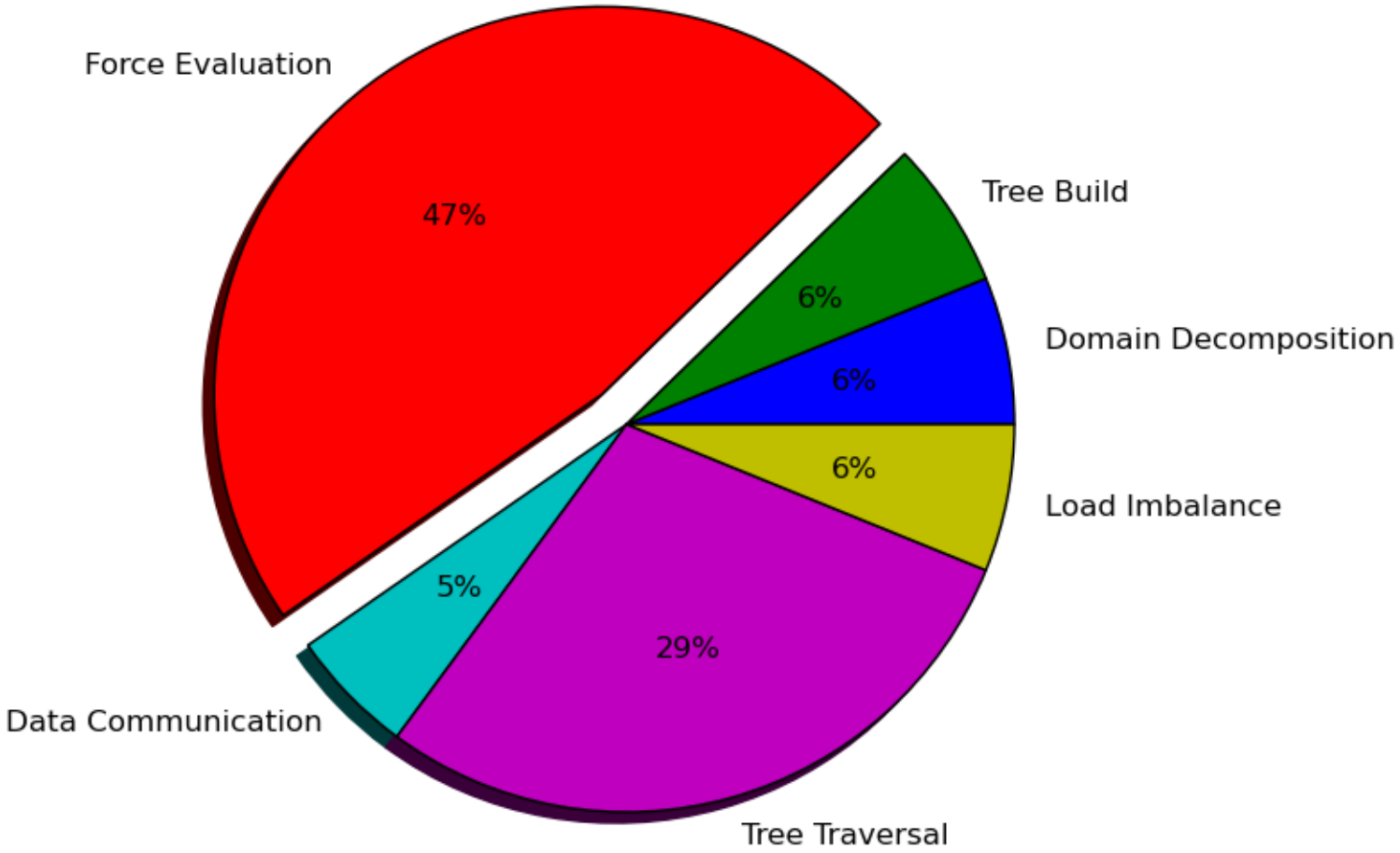


Figure 2: Representative fractions of time spent in phases of parallel code

Can you outrun Moore's Law?

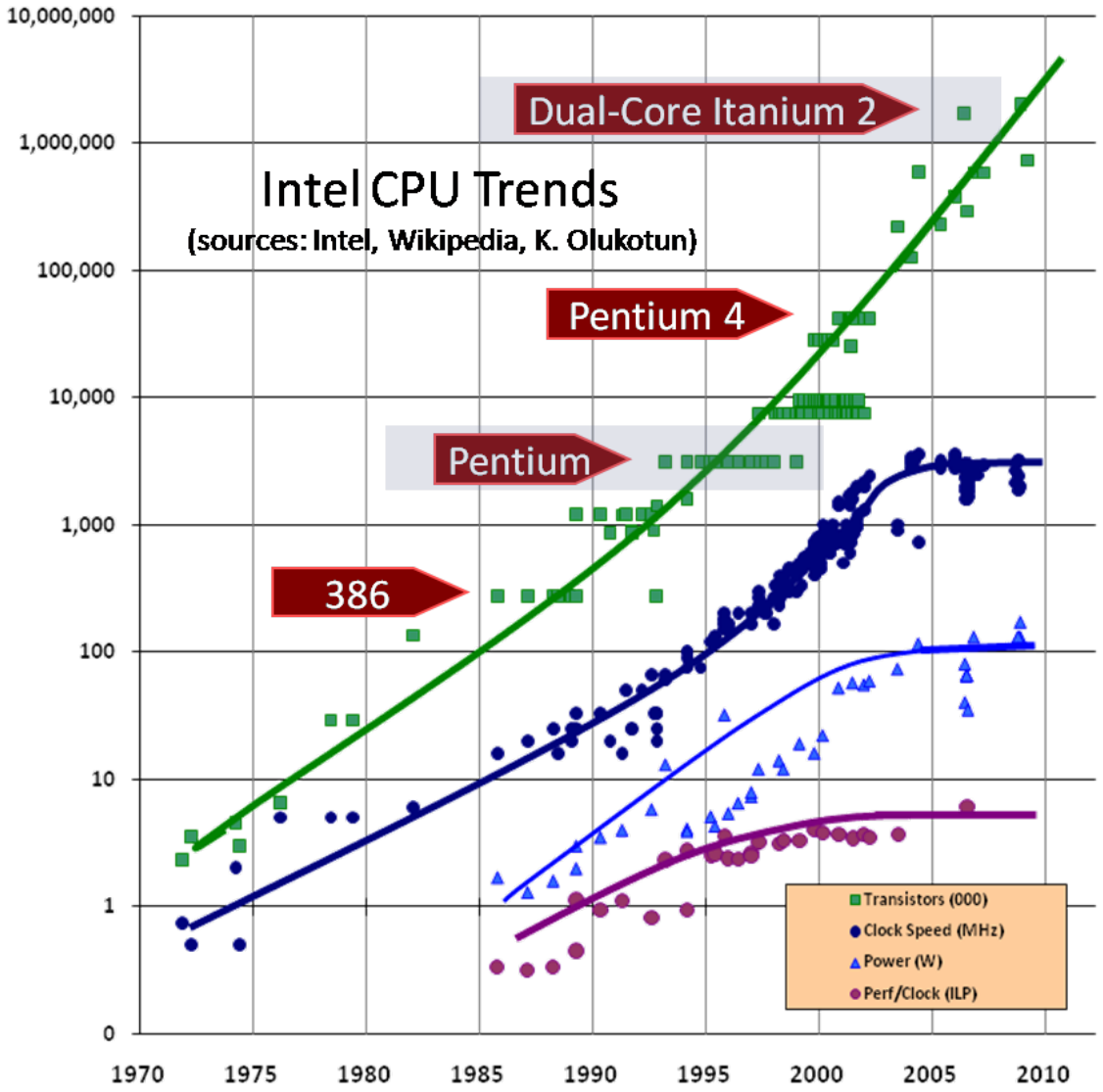


Figure 3: Intel CPU Introductions (Herb Sutter)

Where are we?

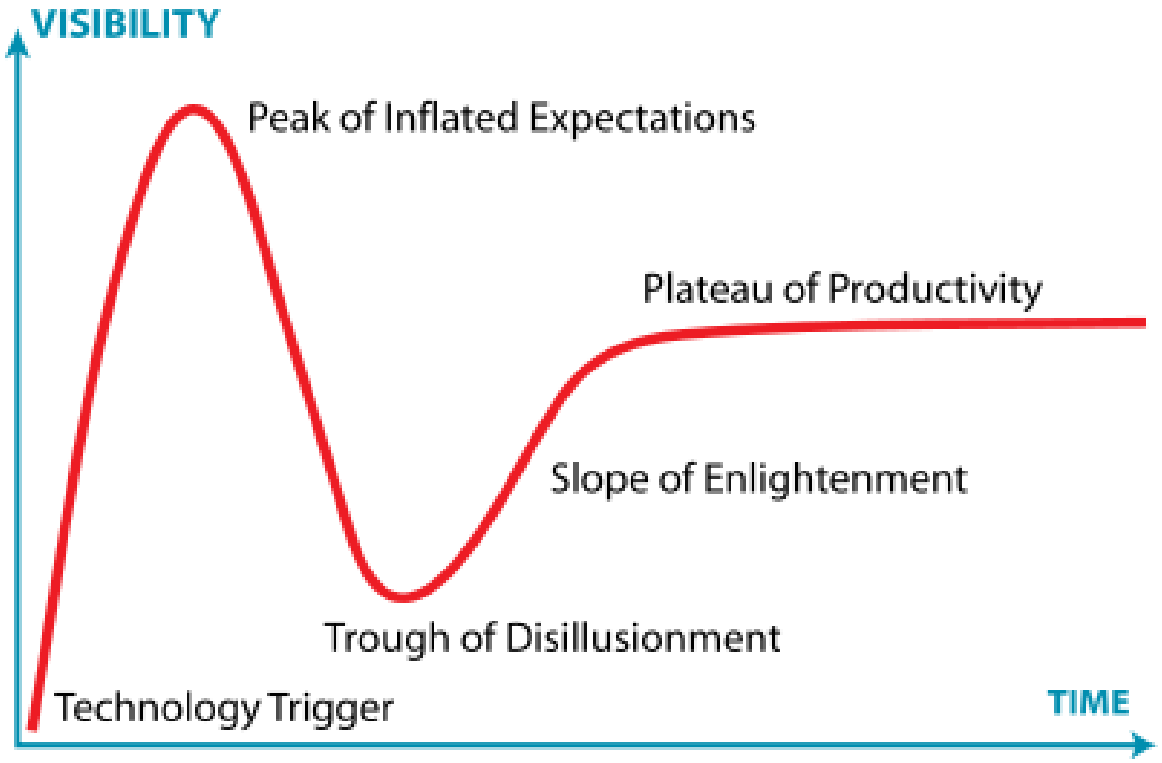


Figure 4: Gartner Hype Cycle

Current CPU Trends

	Pure CPU Driven Systems								
	2009		2010			2011		2013+	
	AMD Istanbul	Intel Nehalem	Intel Westmere	AMD Magny-Cours 4-Socket	Intel Nehalem-EX	AMD InterLagos 4 Socket	Intel Sandy Bridge	Future Intel	Future AMD
FLOPS Per Clock	4	4	4	4	4	4	8	8	8
Clock Speed (GHz)	2.8	2.93	2.93	2.4	2.26	2.3	2.93	2.93	2.4
Number of Cores per Socket	6	4	6	12	8	16	8	16	24
Number of Sockets per Node	2	2	2	4	4	4	2	2	4
Per Node GFLOPS (peak)	134.40	93.76	140.64	441.60	289.28	588.80	375.04	750.88	1,843.20

Figure 5: PetaFLOPS for the Common Man (Jeff Layton, Dell)

Specific Issues

- Vectorization (SIMD)
- Instruction Latency (pipelining)
- Reciprocal Square Root (Newton-Raphson if necessary)
- Memory Bandwidth (Kills co-processor approach)
- Available registers (Limits concurrency)
- Swizzling (Data Layout)
- Portability/Libraries (Learn from BLAS/Automatic Tuning?)

SSE3 Code for Inner Loop

112 flops in 38 cycles, 11 add/sub, 9 mul, 1 rsqrt

```
loop:
  movaps    (%rdi), %xmm0      # mass        rsqrtps   %xmm4, %xmm4
  movaps    16(%rdi), %xmm5    # x           movaps    %xmm0, %xmm1
  movaps    32(%rdi), %xmm6    # y           addps     %xmm1, %xmm11      # mass
  movaps    48(%rdi), %xmm7    # z           addq      $64, %rdi
  subps     %xmm8, %xmm5                     cmpq      %rsi, %rdi
  subps     %xmm9, %xmm6                     mulps     %xmm4, %xmm0
  subps     %xmm10, %xmm7                    mulps     %xmm4, %xmm4
  movaps    %xmm5, %xmm1                     mulps     %xmm0, %xmm4
  movaps    %xmm6, %xmm2                     addps     %xmm0, %xmm12      # phi
  movaps    %xmm7, %xmm3                     mulps     %xmm4, %xmm5
  movaps    (%rsp), %xmm4      # eps         mulps     %xmm4, %xmm6
  mulps     %xmm1, %xmm1                     mulps     %xmm4, %xmm7
  mulps     %xmm2, %xmm2                     addps     %xmm5, %xmm13      # ax
  mulps     %xmm3, %xmm3                     addps     %xmm6, %xmm14      # ay
  addps     %xmm1, %xmm4                     addps     %xmm7, %xmm15      # az
                                             jb        .loop          # Prob 97%
  addps     %xmm2, %xmm4

  addps     %xmm3, %xmm4
```

Specific Issues

- Vectorization (SIMD)
- Instruction Latency (pipelining)
- Reciprocal Square Root (Newton-Raphson if necessary)
- Memory Bandwidth (Kills co-processor approach)
- Available registers (Limits concurrency)
- Swizzling (Data Layout)
- Portability/Libraries (Learn from BLAS/Automatic Tuning?)

Pipelines

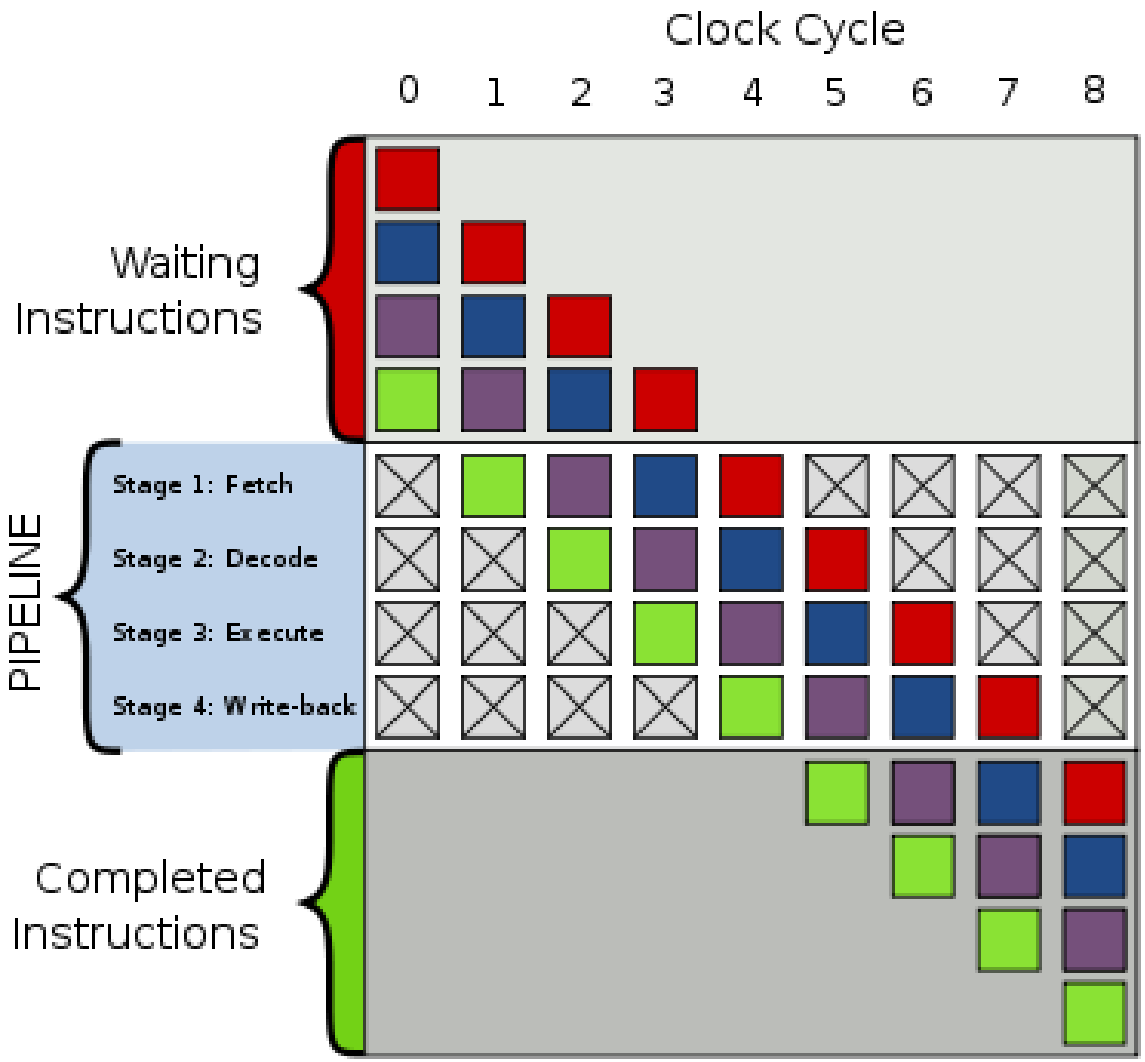


Figure 6: Instruction Pipelining (Wikipedia)

Cell Code for Inner Loop

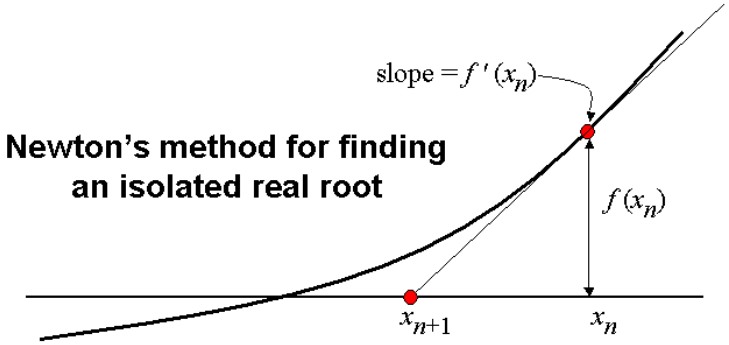
/* Need to unroll by pipeline depth (6) for optimal performance */

```
for (i = 0; i <= n/16; i++) {
    mass = *(vector float *)p;
    dx = *(vector float *)(p+4);
    dy = *(vector float *)(p+8);
    dz = *(vector float *)(p+12);

    dx = spu_sub(dx, pposx);
    dy = spu_sub(dy, pposy);
    dz = spu_sub(dz, pposz);

    dr2 = spu_madd(dx, dx, eps2);
    dr2 = spu_madd(dy, dy, dr2);
    dr2 = spu_madd(dz, dz, dr2);

    phii = spu_rsqrte(dr2);
    mor3 = spu_mul(phii, phii);
    phii = spu_mul(phii, mass);
    total_mass = spu_add(total_mass, mass);
    p += 16;
    mor3 = spu_mul(mor3, phii);

    phi = spu_sub(phi, phii);
    ax = spu_madd(mor3, dx, ax);
    ay = spu_madd(mor3, dy, ay);
    az = spu_madd(mor3, dz, az);
}
```

Specific Issues

- Vectorization (SIMD)
- Instruction Latency (pipelining)
- Reciprocal Square Root (Newton-Raphson if necessary)
- Memory Bandwidth (Kills co-processor approach)
- Available registers (Limits concurrency)
- Swizzling (Data Layout)
- Portability/Libraries (Learn from BLAS/Automatic Tuning?)

i860 Assembly Code



$$x_{n+1} = x_n - \frac{f'(x_n)}{f(x_n)}$$

```
frsqr.ss g0, t0          // initial guess of inverse sqrt
frsqr.ss g1, t1
frsqr.ss g2, t2
pfmul.ss c2, g0, f0      // 0.5 * x
pfmul.ss c2, g1, f0
pfmul.ss c2, g2, f0
pfmul.ss t0, t0, g0      // yold * yold
pfmul.ss t1, t1, g1
pfmul.ss t2, t2, g2
pfmul.ss g0, t3, t3      // (0.5 * x) * (yold * yold)
pfmul.ss g1, t3, t3
pfmul.ss g2, t3, t3
mr2s1.ss  c1, f0, f0     // 1.5 - ( )
mr2s1.ss  c1, f0, f0
mr2s1.ss  c1, f0, f0
pfadd.ss f0, f0, t3
pfmul.ss t0, t3, f0      // yold * ( )
pfadd.ss f0, f0, t3
pfmul.ss t1, t3, f0
pfadd.ss f0, f0, t3
```

```
pfmul.ss t2, t3, f0
pfmul.ss g0, t0, t0      // (0.5 * x) * yold
pfmul.ss g1, t1, t1
pfmul.ss g2, t2, t2
pfmul.ss t0, t3, t3      // yold * ( )
pfmul.ss t1, t3, t3
pfmul.ss t2, t3, t3
mr2s1.ss  c1, f0, f0     // 1.5 - ( )
mr2s1.ss  c1, f0, f0
mr2s1.ss  c1, f0, f0
pfadd.ss f0, f0, t3
pfmul.ss t0, t3, f0      // yold * ( )
pfadd.ss f0, f0, t3
pfmul.ss t1, t3, f0
pfadd.ss f0, f0, t3
pfmul.ss t2, t3, f0
pfmul.ss f0, f0, g0      // result
pfmul.ss f0, f0, g1
pfmul.ss f0, f0, g2
```

Specific Issues

- Vectorization (SIMD)
- Instruction Latency (pipelining)
- Reciprocal Square Root (Newton-Raphson if necessary)
- Memory Bandwidth (Kills co-processor approach)
- Available registers (Limits concurrency)
- Swizzling (Data Layout)
- Portability/Libraries (Learn from BLAS/Automatic Tuning?)

Memory Bandwidth

"Those who cannot remember the past are condemned to repeat it."

—George Santayana

```
/* Copy Data for CM-5 Vector Unit */
bodies = (aux float *)aux_alloc_heap(n);
for (j = 0; j < cnt/(4*VLEN*N_DP); j++) {
   for (i = 0; i < N_DP; i++) {
      dp_copy((double *)(p+(i+j*N_DP)*4*VLEN),
              (double *)(p+(i+1+j*N_DP)*4*VLEN),
              (aux double *)(bodies+(last_icnt*4/N_DP)+j*4*VLEN), dp[i]);
   }
}
bodies = AC_change_dp(bodies, ALL_DPS);
do_grav_vu(bodies, bodies+n/N_DP, pos0, &mass, acc, &phi, eps2p);
```

The gravitational inner loop requires approximately 1 byte/sec of bandwidth for each floating point operation per second. e.g. a Cell QS22 blade requires 256 Gbytes/sec, a Tesla C1060 requires 512 Gbytes/sec. PCI-Express x16 provides 3.2 Gbytes/sec.

Specific Issues

- Vectorization (SIMD)
- Instruction Latency (pipelining)
- Reciprocal Square Root (Newton-Raphson if necessary)
- Memory Bandwidth (Kills co-processor approach)
- Available registers (Limits concurrency)
- Swizzling (Data Layout)
- Portability/Libraries (Learn from BLAS/Automatic Tuning?)

Specific Issues

- Vectorization (SIMD)
- Instruction Latency (pipelining)
- Reciprocal Square Root (Newton-Raphson if necessary)
- Memory Bandwidth (Kills co-processor approach)
- Available registers (Limits concurrency)
- Swizzling (Data Layout)
- Portability/Libraries (Learn from BLAS/Automatic Tuning?)

Specific Issues

- Vectorization (SIMD)
- Instruction Latency (pipelining)
- Reciprocal Square Root (Newton-Raphson if necessary)
- Memory Bandwidth (Kills co-processor approach)
- Available registers (Limits concurrency)
- Swizzling (Data Layout)
- Portability/Libraries (Learn from BLAS/Automatic Tuning?)

Historical Treecode Performance

Year	Site	Machine	Procs	Gflop/s	Mflops/proc
2006	LANL	Coyote (SSE3)	448	1880	4200.0
2004	LANL	Space Simulator (SSE)	288	1166	4050.0
2003	LANL	ASCI QB	3600	2793	775.8
2002	NERSC	IBM SP-3(375/W)	256	57.70	225.0
2000	LANL	SGI Origin 2000	64	13.10	205.0
1996	Sandia	ASCI Red	6800	464.9	68.4
1995	JPL	Cray T3D	256	7.94	31.0
1995	LANL	TMC CM-5	512	14.06	27.5
1993	Caltech	Intel Delta	512	10.02	19.6

Table 2: Performance of HOT on a variety of parallel supercomputers.