# Poor Man's Computing Revisited

**Alexander Shchepetkin, I.G.P.P. UCLA**

**Venice, Italy, October 2004**

a follow-up presentation for **Poor Man's Computing: how much power you can get from a Linux PC,** Seattle, WA, August 2003

## Poor Man's lessons from the past

- Specifics of architecture: strong and weak points
- Software side of the story: availability, performance
- Specific issues of code optimization
- Comparison with SGI ORIGIN 2000, Sun Enterprise

## Computer #1: 2 $\times$ 933 MHz Pentium IIIs on ASUS CUV4X-D (VIA 694XP set), 1024MB PC-133 memory

**hardware**

- SMP-architecture (shared bus like SGI Power Challenge)
- complex instruction set (not a RISC processor)
- 32-bit architecture (addressing space is limited to 4GB)
- 80-bit long registers (!) allowing 32-, 64-, and 80-bit single, double or *extended*-precision arithmetic. In the last case results are truncated to 64 bit, when leaving registers.
- full-speed, 256 KB 8-way (!) set-associative cache
- 1.06 GB/sec memory bandwidth, 64-bit data bus, $\approx 70ns$ latency

**software**

- **multi bootable:** Linux Mandrake 8.2 "OEM Standard" (running 2.4.18-8.1smp.mdk kernel); and Linux Mandrake 7.2 "Complete" (2.2.17smp kernel) operating systems; as well as Windows 2000.
- GNU `gcc`, `g77` compilers (F77 but not F90); *all FREE*
- Intel `icc` and `ifc` compilers: F95 with Open MP support; *FREE for Linux only*
- Lahey (Fujitsu) `lf95` compiler, striped down version (no `OpenMP` support) $240; Complete version $640
- State of the art in early 2001, worthless today

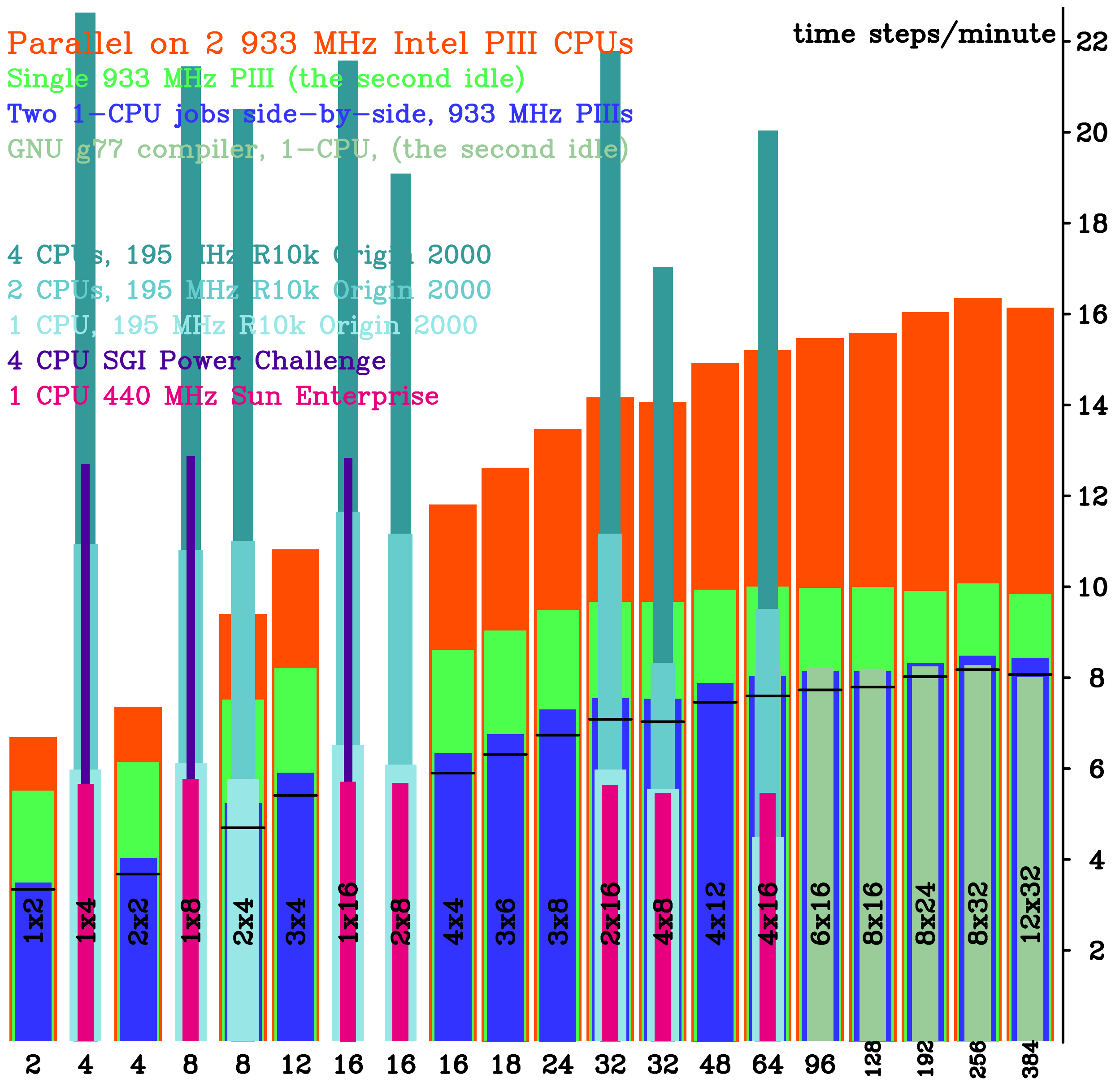
## Test Problem:

**ROMS code** (relevant features):

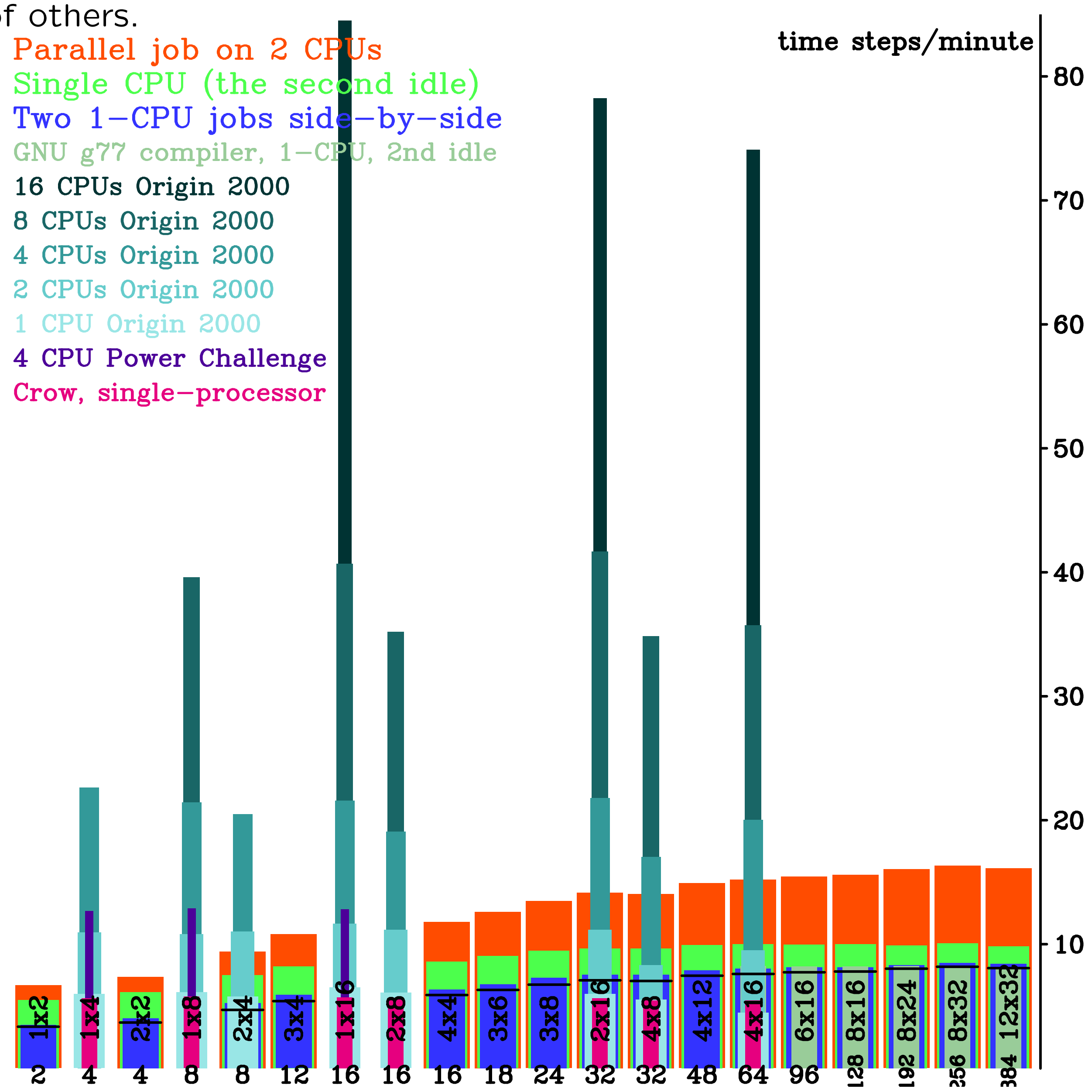
- parallel code using Open MP or MPI, with 2D subdomain decomposition. Originally designed/optimized for SGI Origin 2000, since them ported on many other platforms;
- number of subdomains may be set *independently* from the number of CPUs used in OpenMP mode;
- mild amount of redundant computations occurs at the perimeter of subdomains — typically involving no more than two ghost points.
- in Open MP mode, if there is more than one subdomain (tiles) per processor, each thread sweeps its tiles in *zig-zag* order — a measure designed to avoid/mitigate *false sharing* on Origin 2000, as well as to improve reuse of cached data.
- all numerical features of ROMS

### Model configuration

- 3/4 degree North - Equatorial Atlantic DAMEE configuration, $128\times 128 \times 20$ grid (similar setup to Haidvogel *et al*, 2000) $\Rightarrow$ **100MB problem**



Computational performance of ROMS code as a function of subdomain partitioning (blocking) policy on different hardware platforms for 3/4 degree Atlantic model ($128\times128\times20$ grid, 100MB storage). Horizontal axis represent the number of subdomains, and specific blocking policy is written on each column in $N_X^{\mathrm{SUB}}\times N_Y^{\mathrm{SUB}}$ format (some numbers of subdomains are repeated because they correspond to different policies). Vertical axis—computational performance — time steps per minute of wall clock time. In all cases parallelization is done via OpenMP and no adjustments to the code are made other than choosing different number of subdomains. Strong dependency of computation performance from number of subdomains for Intel platform is explained by cache effects due to combination of small cache, fast processors and limitation by memory bandwidth, which is by far the dominant factor for optimization strategy in this case. For all other platforms the effect is less significant, and, in fact the most significant influence on performance can be traced to the side effects due to shortening of innermost loops when decreasing subdomain size. Nevertheless, for a properly optimized code (number of subdomains is chosen to make subdomains sufficiently small to fit into cache), even the previous generation of Intel platforms tends to outperform the other computers presented here in terms of processing power per CPU, despite the fact that its cost is only a small fraction of the cost of others.



Same as before, but showing cases with up to 16 Origin 2000 CPUs involved: the code scales perfectly to this number of processors and it is clear that it is vector length which matters most for Origin 2000 performance: blocking of the first FORTRAN dimension causes performance degradation, despite the potentially beneficial effect onto cache utilization. Sun Enterprise generally shows much lesser dependency on vector length which may be mostly explained by efficiency of Sun compilers (note 195 MHz R10k of Origin 2000 delivers the same performance as 400 MHz Sun Enterprise)

## Poor Man's experience:

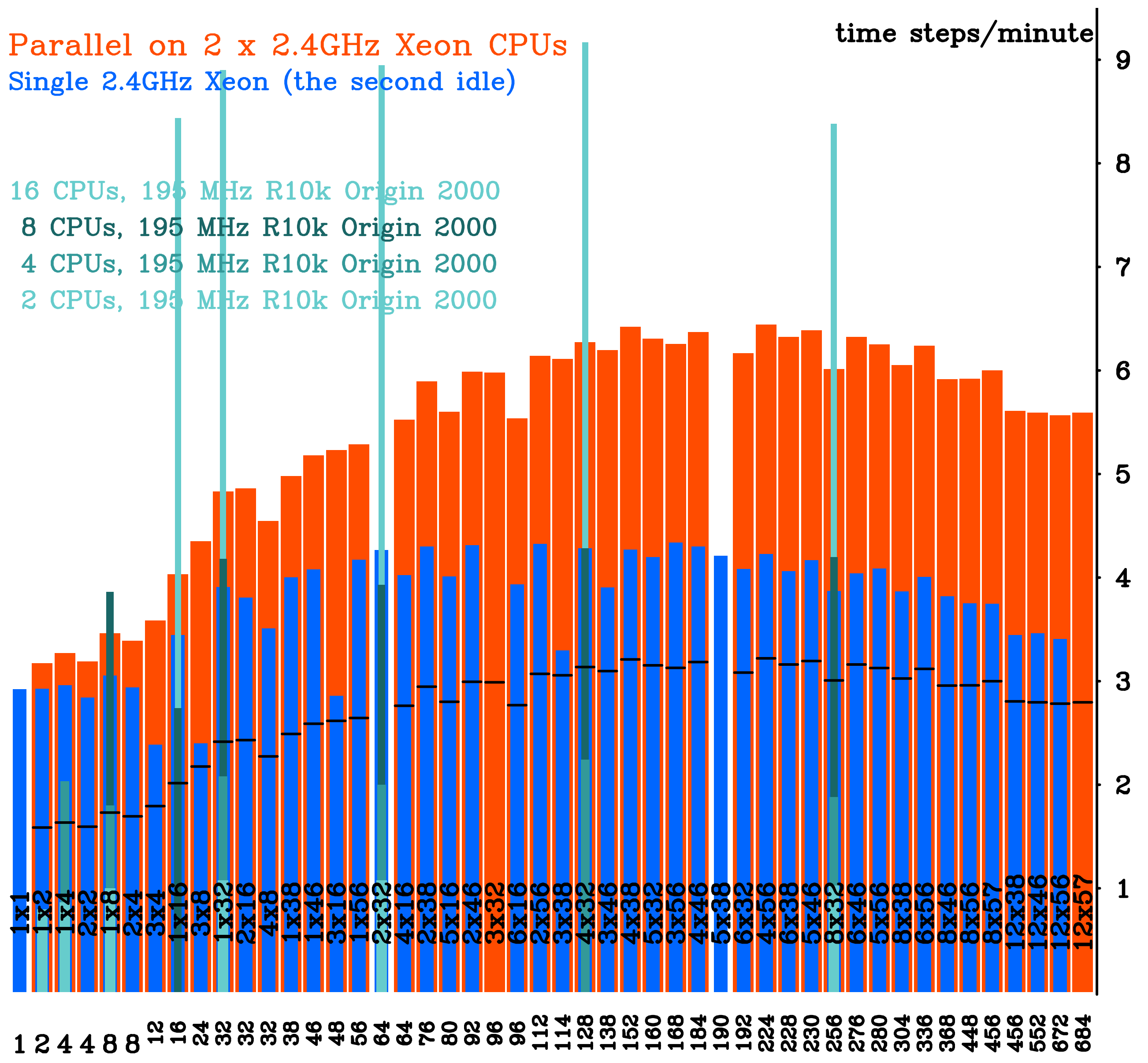
- Linux operating systems were mature and robust at that time. Probably even earlier in 1999. It just took long time to recognize and embrace it.
- Optimization strategies on Pentium PC are significantly different from that for traditional workstations and supercomputers with major accent placed on utilization of cache, and taking into account limited memory bandwidth. Length of innermost loops (i.e., vector loops) become much less important for PIII.
- subdomain partitioning of ROMS with multiple subdomains (tiles) per processors can be used to solve cache management problem: as paradoxically as it may sound, one needs parallel code to run it efficiently on a laptop.
- For a properly optimized code 2 $\times$ 933 MHz PIIIs deliver similar (slightly better) performance as three 195 MHz R10k of Origin 2000.

## Computer #2: 2 $\times$ 2.4 GHz Intel Xeon CPUs on Supermicro P4DCE+ board (i860 chip set), 1024MB memory

- PC800 RDRAM memory $\Rightarrow$ 3.2 GB/sec bandwidth; 40$ns$ latency
- 512 KB cache; 4x cache line relatively to PIII
- Linux Mandrake 9.1 (kernel 2.4.21-024smp/ent)
- Intel `ifc/icc` 6.0.1-304 and 7.1.0xx compilers
- Cost to build $1700 (beginning 2003)

## Test Problem:

ROMS code, same as above;
1/2 degree Pacific model, $384 \times 224 \times 30$ grid, $\Rightarrow$ **800MB+ problem**



Performance 1/2 degree, $384 \times 224 \times 32$ grid Pacific model on dual 2.4 GHz Xeon machine: Overall, with proper choice of partitions the dual Xeon machine runs as fast as to 12 195MHz R10k CPUs of Origin 2000. It takes 10 hours of computing (wall clock) to get one model year of simulation, which makes it viable choice (time step 7200 sec; mode splitting ratio ndtfast=78; FB barotropic mode).

## What did we learn:

- Cache utilization has major effect. Dual-Xeon machine CPUs are in more "data-hungry" situation that PIIIs above: it looks like per-CPU computational speed has increased by a factor of 4+, while its clock speed only in 2.5 and memory bandwidth by a factor of 3.

- With introduction of P4 **vector length is back into consideration again** (P4 has 4× longer cache line; pipelined regime)

- The optimal partitioning corresponds to the best compromise trying to satisfy three demands: (1) storage size corresponding to one subdomain should fit into processor cache to facilitate efficient reuse of cached data. (2) a mild amount of redundant operations takes place near the perimeter of each subdomain, hence blocking refinement *increases* the actual number of computations, bringing the "perimeter vs. area" consideration; (3) Pentium 4 and P4/Xeon processors use pipelined regime to achieve efficient computations, and in addition to that they must load consecutive double-precision numbers in *quads* (in Intel's terminology this is called "vectorization") for efficient memory access. Pipelining require long inter-most loops (our experience shows that these loops must exceed ∼100 iterations to avoid penalties for side effects), and *quad*-loading stipulates that innermost loops correspond to the first Fortran dimension.

- Compromise is *very compromising*: none of the three requirements is anywhere close to satisfaction. Scaling is no better than dual-PIII.

- Best policy results subdomain size of 96×4 points — **try MPI code with 4-point wide subdomains + 2 ghost points on each side!**

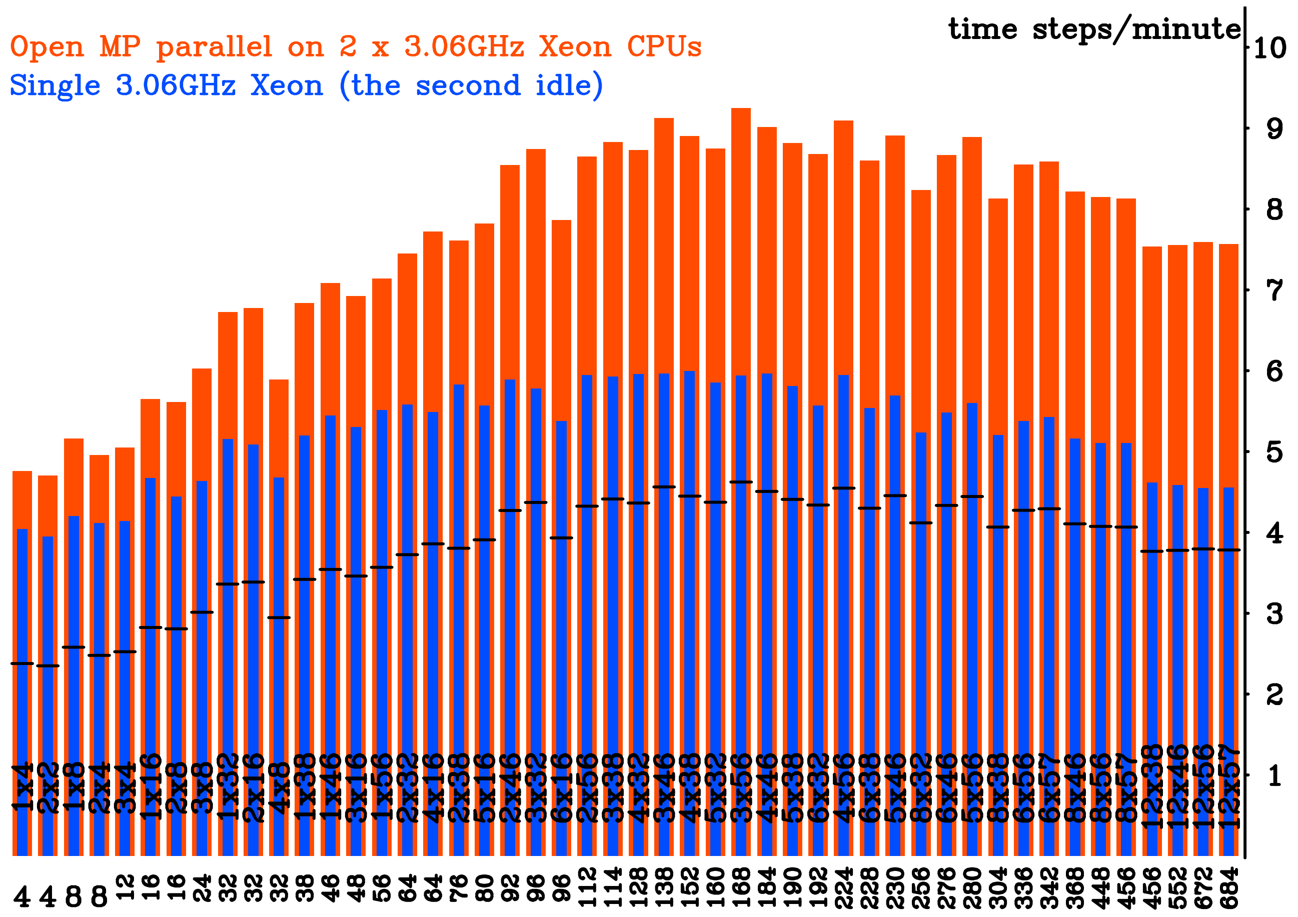
- Overall, for a properly optimized code Dual 2.4GHz Xeon runs as fast as 12 × R10k's.

## Poor Man's Computing Today

**Computer #3:** NCSA Linux Cluster made of dual 3.06GHz Xeon Dell PowerEdge 1750 nodes connected via Myranet network. This cluster is also known as *tungsten*.

running Red Hat Enterprise Linux, kernel 2.4.20-31.9smp. Intel Fortran compiler 7.1 and **8.0** (make sure release 8.0.039 or later, earlier versions of 8.0.x are known to miss-interpret `OMP$ THREADPRIVATE` (!)).

[This work was done *before* July-august 2004 upgrade of NCSA Xeon Cluster, hence all results reported here are a obtained using 3.06GHz CPUs with 512K L2 caches and 533MHz front bus speed.]

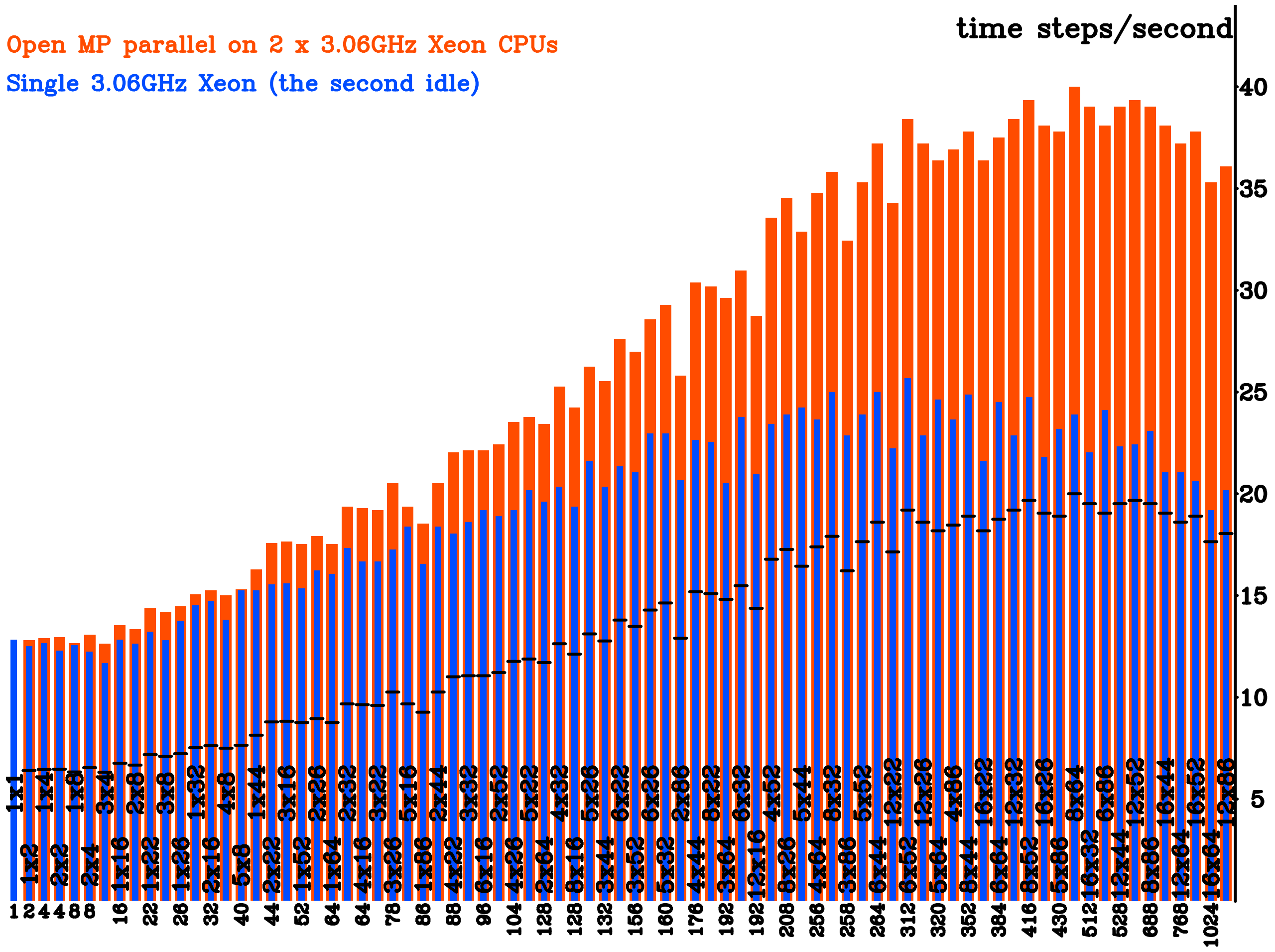


Performance of ROMS code on a Dell PowerEdge 1750 server with two Intel Xeon 3.06GHz/512Kb cache processors using one (the second kept idle) to both CPUs. The test problem is ROMS 1/2-degree Pacific Ocean model on $384 \times 224 \times 30$ (the same as above).

Compared with our home-made dual-Xeon machine, the PowerEdge 1750 node has 1.25 times faster CPU clock speed (3.06 vs 2.5); 4/3 times more memory bandwidth due to 533 Mhz front bus; and **2.1 times better memory latency** (CAS 2.5 at 133 MHz $\Rightarrow$ 18.75$ns$ vs. 40 $ns$).

**Results are not fundamentally different:** as before, there is a strong dependency of computational performance (as well as scaling from 1 to 2 CPUs) from the number of subdomains and blocking. It is $\approx$ 30% faster, however optimum partitioning leans to slightly longer vector length: $3 \times 56$. Most likely this is attributed to significantly smaller memory latency.

Given that the total storage size for this problem is 850 MBytes, an intuitive "rule of thumb" tells that 1700 subdomains are needed to fit into 512Kb cache. In practice the number is **significantly smaller** — about 160. This is because not all model arrays are being used in each parallel region, and, more importantly, the best effort is already made to organize loops within subdomains in cache-friendly manner: the operations are arranged sweeping either in horizontal $x-y$ or vertical $x-z$ planes with intermediate results placed in two-dimensional (almost never three-dimensional) tile-sized scratch arrays, so that successful cache management is achieved when a sufficient number of these rather small size scratch arrays fit into cache.



To illustrate the significance of cache blocking within the *internal code*, we repeat a similar experiments, but with a 2D problem, where all cache management relies entirely on subdomain decomposition. This is a $768 \times 256$-grid two-dimensional Soliton problem run for 4800 time steps.

The behavior is overall similar to the 3D, but with larger contrast coarse- and fine blocking, especially in dual-CPU case. This problem uses total of 57 MBytes of memory, and unlike in 3D case, the optimum blocking $8 \times 64$ results in $\approx$ 115Kb per subdomain, which entirely fits into cache. The best subdomain dimension $\sim 100 \times 4$ coincides with that for 3D problem.

## Cluster Computing

### It has to be MPI...

Scaling performance of ROMS code on NCSA Xeon cluster. The same 1/2-degree Pacific Ocean model on $384 \times 224 \times 30$ grid, and the duration of each run is 512 time steps. All these test runs are obtained using purely MPI-mode.

| **number of CPUs** | **4** | **8** | **12** | **16** | **24** |
|---|---|---|---|---|---|
| partition | 2×2 | 2×4 | 3×4 | 4×4 | 3×8 |
| **run time**, *seconds* | **4130** | **2154** | **1118** | **868** | **521** |
| time steps/*minute* | 7.44 | 14.2 | 27.5 | 35.4 | 58.9 |
| time steps/*minute*/node* | 3.71 | 3.57 | 4.57 | 4.42 | 4.91 |

| **number of CPUs** | **32** | **48** | **64** | **96** |
|---|---|---|---|---|
| partition | 4×8 | 3×16 | 8×8 | 6×16 |
| **run time**, *seconds* | **413** | **241** | **173** | **139** |
| time steps/*minute* | 74.4 | 127.5 | 177.5 | 221.0 |
| time steps/*minute*/node* | 4.65 | **5.29** | **5.53** | 4.60 |

Term *node* labeled by asterisk * in the last line means the whole *dual-CPU* PowerEdge 1750 node, hence *node*=2CPUs

I/O during these runs is done from/into separate files *individually* for each MPI node using the standard (non-parallel) netCDF library. This leads to scalable I/O, however pre- and post-processing is required to convert the data into more a conventional form.

The machinery and the code exhibit overall good scaling, which is manifested nearly proportional decrease of run time with the number of CPUs, as well as non-degrading per-node computational performance — shown on the bottom line. However the per-node performance above is **significantly lower** than the 8.9 time steps/minute of the properly optimized Open MP code running on 2 CPUs (just one node of the Linux cluster) using fine $3 \times 56$ partitioning to better utilize processor's cache. Mild super-linear scaling and partial recovery of per-node performance on 48 and 64 CPUs above (bottom line, highlighted) is attributed to cache effects due to effective reduction of problem size solved on each node.

- Scalable parallel I/O in imperative

## Conclusion

- Since 1997, when Origin 2000 was introduced, clock speed workstation-class computers increased by a factor of 5 or so (200 MHz R10k → 800 MHz R14k for SGI; and 330 MHz UltraSparkIIi then → 900 MHz today for Sun), while performance of PC hardware has been increased more that 15 times
- Although memory bandwidth of PCs has been improved by a factor of 10 in last five years — 500...666MB/sec of PC66...PC100 in 1999 → 1.06 GB/sec of PC133 (mainstream PIII generation PCs of year 2000) → 2.1 GB/sec of DDR PC2100 → 3.2 GB/sec RDRAM of first P4 → 6.4 GB/sec of dual-channel DDR (Intel i875/i865 sets) of today — still, modern PC are even more *miss-balanced* than their predecessors.
- SMP designs double processor power, but memory bandwidth remains the same, which limits scalability.
- **Xeon gap**: P4/Xeon loses to ordinary P4 front bus speed. Recently and today it was/is 533 vs. 800; a couple of month ago a 800MHz version of Xeon "Nocona" has been released, but regular P4 is heading toward 1066. **This exacerbates the miss-balance** to roughly 3:1 (ouch!) of dual-Xeon vs. single-P4 ability to perform arithmetic operations per load-stores.
- Memory interleaving (common design to boost bandwidth for supercomputers) appears in modern commodity PCs in form of dual-channel RDRAM (2001 i850/i860 chip sets) and dual-channel DDR of year 2003 (i875/i865, E750x, as well as SerwerWorks sets). There is only 2-way interleaved systems on the market today).
- Single 3.2GHz/800MHz front bus/1M cache P4 + i875P (say an Intel D875PBZ board) + dual-channel DDR (must be CAS 2(!)) memory **matches** performance our home-made dual-Xeon machine.
- In our experience Intel's *hyper-threading* (R,TM) technology does not give any advantage for parallel computing: whether it in dual-Xeon or single-P4, performance was always better is that thing was turned off in BIOS. Nevertheless, Intel's news about *dual-core* CPUs (real ones, not fake) is quite exiting.
- Cost-performance consideration favors single-processor PCs for Linux clusters

- A typical MPI code with one subdomain — one processor strategy is out of cache and would not perform/scale well on a PC hardware, dual- or single-processor

- Similar cache effects were observed on IBM p690 (1.3GHz Power4 CPUs with shared 3 level caches — turns out that fitting into L2 cache is of the most importance), SGI Altix (Intel Itanium 2 CPUs, 64-bit Intel EFC compiler), AMD Opteron (2 GHz, 1M L2, Intel IFC/Portland Group PGF compilers for 32-/64-bit respectively), and MacIntosh G5 (2 GHz, 1M L2,IBM XLF compiler). In four cases blocking into $\sim$ 120...20$\times$4-grid point tiles shows the optimum, with 64-bit machines care a bit more about vector length.

- Intuitive *rule of thumb* — the total memory divided by the number of subdomains should fit into L2 cache — does not provide an accurate guideline for today's ROMS code. It is obscured by heavy usage of private scratch arrays [4 3D and up to 27 2D arrays (27 in `visc3D_GP` and 16 `step2D_FB` routines)].

- Comparison of ROMS performance Open MP and MPI modes indicates that there is underutilization of internal potential of CPU in current implementation of MPI code. This is not a principal inherent drawback of MPI parallelization framework, but rather due to the fact that current implementation sticks to single-subdomain — single CPU policy in MPI, while our current Open MP implementation is free of this restriction. **This needs to be addressed**.

## Afterthoughts

..At the early days of ocean modeling the computers were too small to hold the whole problem in memory, so the codes were designed to run using disk as the primary storage. Memory was fast but small, and disk is much larger, but is also much slower, so coding was an art of making as much computing as possible before a new portion of data needs to read from the disk. Now history came full cycle, except that what used to be memory now is called *cache*, and what used to be disk now became *memory*. The art remains.

...Today's supercomputer centers have most of their power in clusters and getting more (80% at NCSA today). Does it look like Poor Man's experience becomes that of Rich Man too?