Benchmarking NEMO on Various Platforms

Benchmarking NEMO on Various Platforms

1 Abstract

The benchmarking of the Nanoelectronic Modeling Tool (NEMO) on a variety of different computational platforms is presented. Three platforms are evaluated as possible development platforms for continued NEMO work. These platforms are SUN E450 Ultra-Sparc 2 (300 MHz), SGI Origin 2000 (200 MHz), and HP V class 8000 (200 MHz). The benchmarking issues to be addressed are 1) single CPU and 2) shared memory multiple CPU performance on a wide variety of physical models, and 3) compilation speeds. Several compiler optimizations were examined on the individual platforms and tested against the standard NEMO binary, which is provided by Raytheon. Several other platforms available to the High Performance Computing Group (HPC) have been benchmarked as well: HP/Convex SPP-2000 (180 MHz), SGI Onyx (200 MHz), LINUX based Pentium II (200 MHz), SUN Sparc-Ultra 1 (170 MHz). These machines are tested as possible execution platforms for NEMO.

2 Background

2.1 What is NEMO?

NEMO is the Nanoelectronic Modeling tool developed at the Applied Research Laboratory of Raytheon, formerly known as the Central Research Laboratory of Texas Instruments. NEMO enables the quantum mechanical simulation of the electron transport through semiconductor heterostructures that contain material modifications in one dimension. NEMO can be accessed through a graphical user interface (GUI) which based on X11/Motif and through a batch run interface. Further details on NEMO can be found at the NEMO home page at http://www.raytheon.com/rtis/nemo/ and the home page of Gerhard Klimeck at http://www-hpc.jpl.nasa.gov/PEP/gekco/.

2.2 What are the JPL Plans?

JPL has acquired a license to the NEMO source code and plans to use the code for the simulation of optical detectors like QWIPs and heterostructure based lasers. Such simulations will require the extensive enhancements of the NEMO code to include strong scattering mechanisms including bandstructure effects and the interaction of the electrons with photons. For this purpose we are planning to purchase a development platform that enables fast source code development and fast debugging.

2.3 NEMO Limitations - Motivation of Benchmarks

The simulation of electron transport in optical devices involves the modeling of electrons that are highly excited above the band edges (bandstructure effects) and the modeling of electrons that are subject to scattering by other electrons, photons, phonons, interfaces, impurities etc..

The NEMO code in its present form can treat scattering from polar optical phonons, interface roughness, alloy disorder and acoustic phonons. The first two scattering mechanisms have been found to have the largest influence on the valley current of resonant tunneling diodes (RTDs). In RTDs the electron transport is typically not scattering dominated. However, electron transport through multiple quantum well structures or superlattices can be scattering dominated. Although NEMO is possibly the most advanced quantum mechanics based device simulator built to date, the present implementation is not capable of simulating a scattering dominated device like a resistor on a fully quantum mechanical basis. The major reason of this limitation is the number of included scattering events and the exclusion of electron-electron (e-e) scattering from the simulation. The simulation of e-e scattering involves interactions that are non-local in space (full matrices) and non-local in energy (inelastic coupling of different total energy channels). At this time NEMO can simulate an infinite or finite number of local (diagonal matrices), elastic (no total energy coupling) scattering events and *one* non-local (full matrix), inelastic (3 energies are coupled) scattering event. One target of this work is to benchmark the existing scattering code with various different scattering model approximations.

Another limitation to the existing NEMO scattering code is the exclusion of bandstructure effects from the scattering simulation, however, NEMO can simulate bandstructure effects without any scattering. Bandstructure effects are treated within an orbital based tight binding representation. Each atomic site is described by a set of interacting orbitals (s, p_x, p_y, and p_z) whose interactions mimic the dispersion of the material. These orbitals are represented mathematically by spatially coupled NxN matrices, where N is the number of orbitals (including spin).

The Benchmarks are aimed at looking at the performance of the no-scattering multiple band models and the scattering single band models. Future developments of the NEMO code will combine the scattering and bandstructure feature and increase the computational demand significantly.

3 The Benchmarks

3.1 Targets and Timing Description

The following types of benchmarks were considered for the determination of the choice of development platform:

1. Single CPU performance - code compiled for highest performance: The NEMO code was run on various platforms compiled in various forms. Typically there are three forms: 1) Std: the standard Raytheon binary code distribution for that platform, 2) native: the newly compiled code on that particular platform. 3) native parallelized: newly compiled code with the automatic parallelization options turned on. The results are summarized in Section 4.1, while details are deferred to Section 8.3
2. Single CPU performance - code compiled for debugging. To check the execution speed of code which can be debugged the executables and all their objects were rebuilt with the "-g" option. This produces code that is instrumented with references to source code lines and optimizations are turned off. The code typically grows by a factor of 2-3 in size and runs significantly slower. During code development this is the code that is run and tested and fast turnaround times are essential to eliminate bugs fast. The results are summarized in Section 4.2.
3. Multiple CPU, shared memory performance based on automatic, compiler based parallelization. Where available the NEMO code was instrumented for parallel execution using the compiler based automatic parallelization. The results are discussed in summarized in Section 4.3 and detailed in Section 8.1. Section 8.2 discusses the parallelization for more than 4 CPUs on a limited number of platforms.
4. Single CPU compilation time for the batch and the GUI code. The NEMO code has presently about 250,000 lines of C, FORTRAN77 and F90 code in about 570 source files. There are two levels of code development: 1) detailed numerical models and 2) code infrastructure. The numerical models are cradled by the code infrastructure in an object-oriented fashion. While this object oriented approach enables code maintenance and fast prototyping, it does require frequent recompilations of the entire source code, when a significant structural header has been changed. A good development platform supports compilers that not only produce code that executes fast but also compilers that themselves run fast. Compile times on the various platforms range from 2-60 minutes. Compilation of the code in parallel on multiple CPUs scales directly with the number of CPUs except for the linking step at the end. The results are summarized in Section 4.4.

The timing was performed by the UNIX "time" command and that time measure was compared to the NEMO built-in time measure. Reasonable agreement between the two different time measures was achieved on all platforms.

The data is typically presented in a raw data form is in units of seconds of run time and in a normalized form. The normalization is computed in the following fashion:

$$N = \begin{cases} \left(\dfrac{T}{T_{norm}} - 1\right) * 100 & \text{if} \quad T_{norm} \le T \\ \left(\dfrac{T_{norm}}{T} - 1\right) * 100 & \text{if} \quad T_{norm} > T \end{cases}$$

A value of +100% corresponds to an execution time of twice the normalization standard. Conversely a value of -100% corresponds to an execution time of one half of the normalization standard. The normalization standard is typically chosen to be the fastest complete set.

3.2 Platforms

As development platform we consider 3 different machines, all of them are shared memory architectures with 4 CPUs:

1. Sun E450 Ultra-Sparc 2 running at 300 MHz.
2. SGI Origin 2000 running at 200 MHz, These machines are presently shipped at a 250 MHz rating and this increased CPU speed has been verified on NEMO code to translate into a execution time reduction to 200/250 = 80%. The actual machine tested is configured with 16 200MHz CPUs and 8 250 MHz CPUs.
3. HP V class PA 8000 at 200 MHz (these machines can be shipped with 233 MHz CPUs).

Other platforms available to the High Performance Computing Group were tested as well. These platforms were tested as possible execution platforms or temporary development platforms for NEMO.

1. HP/Convex SPP-2000 - 256 CPUs at 180 MHz.
2. SGI Onyx - 4 CPUs at 200 MHz (this is the old style Onyx machine, not Onyx 2, which is presently shipped).
3. Intel Pentium II - 16 CPUs at 200 MHz running the LINUX operating system. This is a non-shared memory machine where parallelization can be achieved using the message passing interface (MPI) and code modifications. The NEMO code is presently not instrumented to use MPI.
4. SUN Ultra-Sparc 1 single CPU at 170 MHz.

3.3 Description of the Physical Models

We ran 9 different input decks for the NEMO benchmarking on the different platform. These benchmarks are briefly described in Table 1. All operations are typically executed in double precision complex numbers.

Abbr.	Max Mem (MB)	Dominant Numerical Operations	Physical Content
sp3s	4	Partial inversion of block-tridiagonal matrices (5x5) with 444 sites.	10 band nearest neighbor tight binding model without explicit treatment of spin. 25 bias points
sp3s_2n	4.5	Partial inversion of block-tridiagonal matrices (20x20) with 222 sites.	10 band second nearest neighbor tight binding model without the explicit treatment of spin. 25 bias points.
sps3_s	6	Partial inversion of block-tridiagonal matrices (10x10) with 444 sites.	10 band nearest neighbor tight binding model with explicit treatment of spin. 25 bias points.
SCB_no_p	4.2	Iterative solution of a system of integral equations. Diagonal matrices with 54 sites. 200 energies are coupled directly, 84 total energies are decoupled and summed up independently.	Infinite number of elastic scattering events. 1 bias point
MSS_no_p	14.1	Recursive solution of a system of integral equations. Diagonal matrices with 54 sites. 200 energies are coupled directly, 84 total energies are decoupled and summed up independently.	Finite number (5) of elastic scattering events. 1 bias point
MSS_d_p	46.6	Same as MSS_no_p, but executed at three different energies at a time. 3 total energies coupled by a diagonal interaction.	Finite number (5) of elastic events and one inelastic event. Inelastic event is approximated as diagonal/local.
MSS_f_p	87	Same as MSS_d_p but full matrix interaction instead of the diagonal matrix interaction.	Same as MSS_d_pop, but eliminating the local interaction approximation.
MSS_f_p_l2	280	Same as MSS_f_p but larger number of spatial sites (76) and larger number of coupled energies (360). Number of uncoupled total energies reduced to 40.	Same as MSS_f_pop, however larger interaction region and better resolution of scattering coupling of the energies
MSS_f_p_l	387	Same as MSS_f_p_l2 but increased number of coupled energies (460) and increased number of decoupled total energies (60).	Same as MSS_f_pop_l2 but increased energy resolution.

Table 1: Benchmarks with their label, maximum memory usage, description of dominant numerical operations and a description of the physical content.

4 The Bottom Line for the Development Platform

The three development platforms we considered for a purchase are an SGI Origin 200 (200 MHz), SUN E450 (300 MHz) and HP V Class 8000 (200 MHz) with 4 CPU's each. The following four sections contain the best data for the four different benchmarks types discussed above. Details of the individual platforms such as runs for different compiler options and the list of compiler options are deferred to Section 8.3.

4.1 Fastest Single CPU Performance

The benchmarks for the fastest execution on these three platforms are presented in Fig. 1. The SGI platform outperforms the other two contenders in most of the benchmark points only to be beaten by the HP in two points by less than 20%.

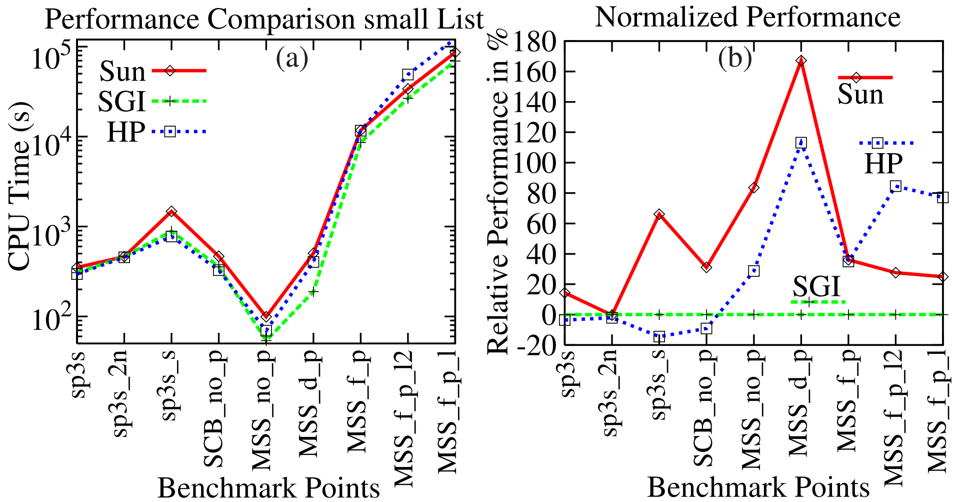

Fig. 1: Fastest single CPU benchmarks performed on the SUN, SGI and HP platform considered for purchase. (b) Contains the data of (a) normalized to the SGI data.

4.2 Debugging Single CPU Performance

The benchmarks for the performance of the code compiled with the debugging option "-g" are shown in Fig. 2. The two extremely time intensive benchmarks MSS_f_p_l2 and MSS_f_p_l are scaled-up versions of the MSS_f_p benchmark and they have not been benchmarked here, since they would not be debugged in that form. The three development platforms produce codes that show significant differences in the execution times. The SUN in particular shows a performance that is 300-750% larger in execution time than the SGI for the multiband benchmarks (sp3s...).

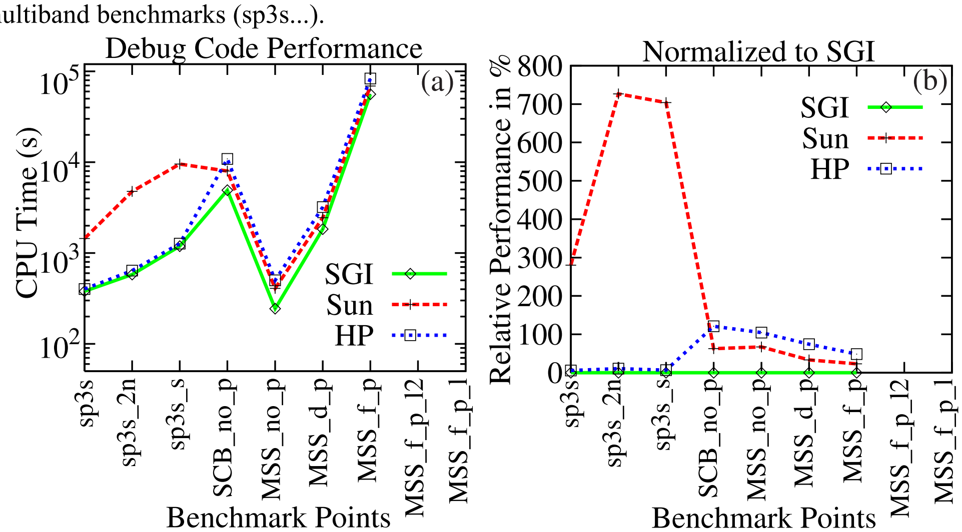

Fig. 2: Benchmarks performed on the SUN, SGI and HP platform considered for purchase with code instrumented for debugging (-g option). (b) Contains the data of (a) normalized to the SGI data.

To visualize the difference in performance of the debugging code and the optimized code we show the relative performance for all the machines normalized to their respective optimized performance in Fig. 3. The debuggable code runs typically 500% (factor 6) slower than the optimized code. The multiband benchmarks (sp3…) show for the SGI and the HP only a small cost for using debuggable code (30-40%). These benchmarks deal with small matrices which fit into cache and no data prefetch optimization is really necessary. The SUN, however, has a high penalty on the debugging form of the executable like the other benchmarks.

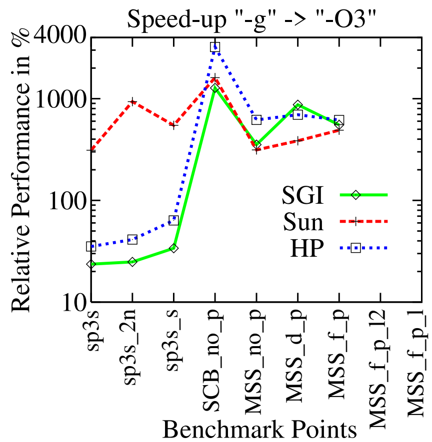

Fig. 3: Speed-up in % due to compilation with optimizations (like -O3) turned on compared to the debugging code performance (option -g).

4.3 Parallelization Comparison

We analyze the efficiency of the automatic parallelization of full matrix manipulation problems. The dominant numerical operation is the LU decomposition and back-substitution of full matrices. These problems appear to be the only ones in the NEMO code, where compiler-based parallelization appears to be effective (for a comparison for all the benchmarks see Section 8.1 below). Fig. 4 presents the raw CPU data, the normalized CPU time and the CPU utilization as a function of employed CPU's in the MSS_f_p benchmark. The CPU utilization for the HP and the SUN platform is better than the SGI utilization, however the SGI delivers the highest throughput still (except for 4 CPU's where the HP wins by less than 5%).

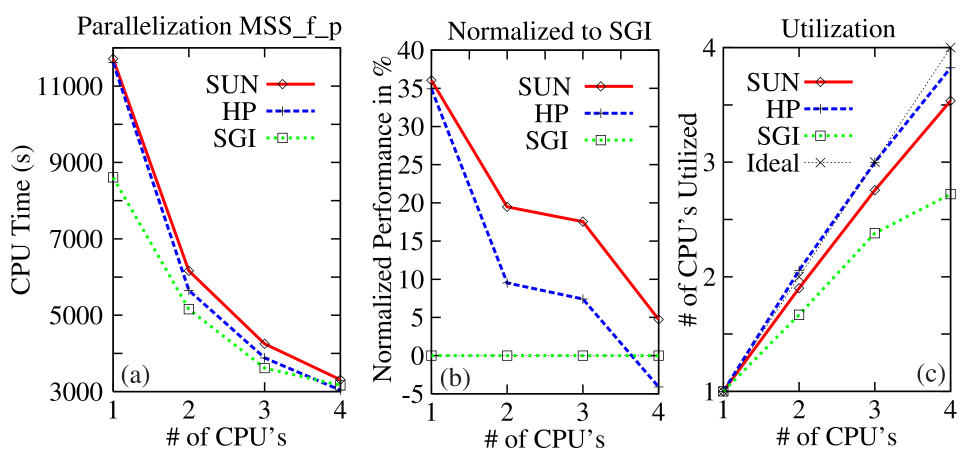

Fig. 4: Parallelization of the MSS_f_p benchmark point. (a) Raw CPU data as a function of number of CPU's. (b) Performance normalized to the SGI data. (c) CPU utilization as function of number of CPU's.

With larger matrices the parallelization efficiency is expected to increase. The "MSS_f_p_l2" benchmark requires significantly more memory compared to the "MSS_f_p" benchmark discussed in Fig. 4 (285 MB vs. 85MB) and the results are shown in Fig. 5.

A serious problem occurred for this benchmark with the SGI Origin 2000. The NEMO code would result in a segmentation fault after several minutes of execution if more than 2 threads for parallel execution were requested. According to an SGI representative this is due to an improper setting of some limits (stack or memory). The problem has not been resolved yet. Although the SGI results in the fastest execution for

1 and 2 CPUs, we cannot use its performance as normalization and choose the next best complete set (SUN).

While the SGI outperforms its competitors on the MSS_f_p benchmark it appears to be loosing its advantage over the other two contenders in this benchmark.

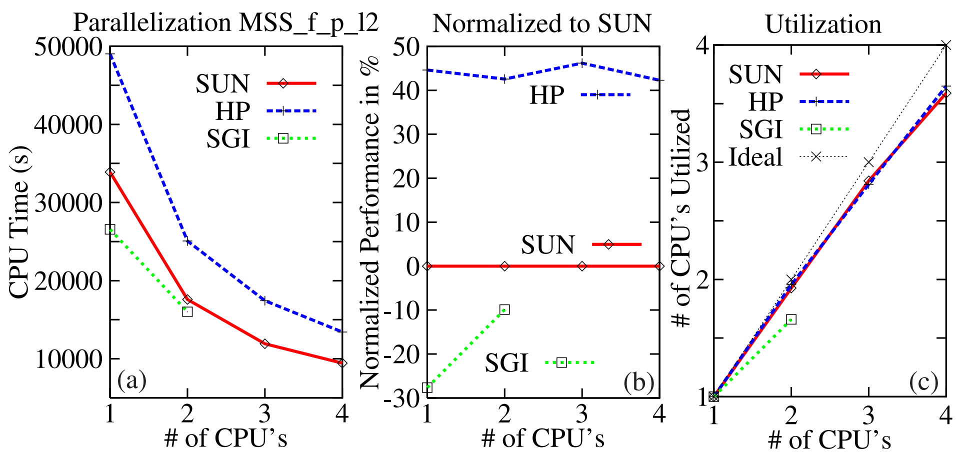

Fig. 5: Parallelization of the MSS_f_p_l2 benchmark point. (a) Raw CPU data as a function of number of CPU's. (b) Performance normalized to the SGI data. (c) CPU utilization as function of number of CPU's.

4.4 Compilation Times

The compilation times of the source code on the three different development platforms are presented in this section. The NEMO software contains about 570 files with about 250,000 lines of code. The batch code does not include about 80 files of C code. The GUI code does include these 80 files and includes a number of X11, Xm, Xt, and XRT related header files in at least 50% of the C files. This inclusion of X heasder files and the additional 40 C and 40 header files slows down the compilation of the GUI code versus the batch code. The compilation time for four different compilation targets has been measured:

1. batch code with debug options, without optimization,
2. batch code with optimization for fastest execution,
3. graphical user interface code (GUI) with debug options, without optimization, and
4. GUI code with optimization for fastest execution.

The SGI outperforms the other two platforms by typically a factor of at least 2. Note that the compilation times for the GUI code are nontrivial of the order of 10-30 minutes. Code will have to be fully recompiled whenever a significant datastructure is changed during code development; large compilation times hinder the code development severely.

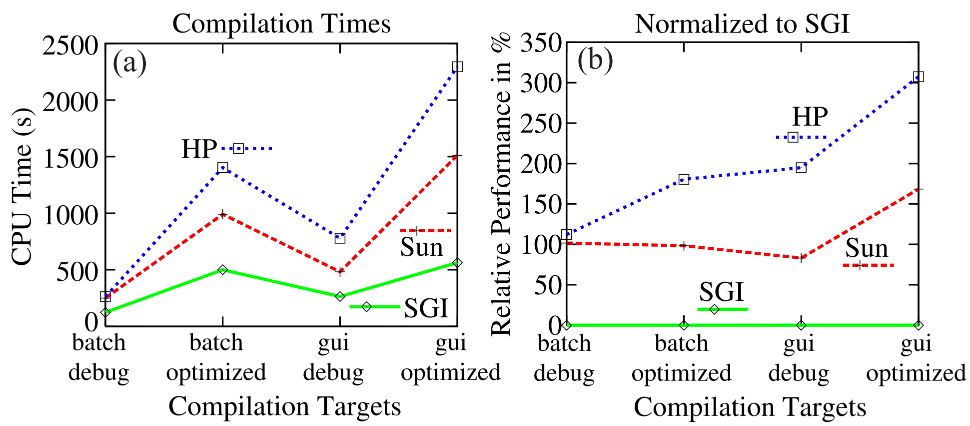

Fig. 6: Compilation times for four different NEMO targets: 1) batch code for debugging, 2) batch code with optimization for fastest execution, 3) graphical user interface code (GUI) for debugging, and 4) GUI code with optimization for fastest execution.

5 Comparison of All Platforms

In addition to the previously discussed SGI Origin 2000, HP V Class and Sun E450 we also benchmarked NEMO on four other platforms available to HPC as listed in Section 3.2. The results are depicted in Fig. 7.

The only candidate to run NEMO for production runs is the HP SPP-2000. Its performance is generally slower on the small matrix problems, however it does perform extremely well in the full matrix MSS_f_p benchmark. The SPP-2000 is a massively parallel machine with 256 CPUs and its parallelization performance is documented in Section 8.2.

The benchmarks on the LINUX based Pentium II were limited to the scattering code due to some complications with the f90 compiler. The full matrix problem with small memory requirements (MSS_f_p) does execute on the LINUX platform with competitive speed. However the machine cannot be used in shared memory mode where large memory problems (> 200MB) are tackled, without code modifications.

The old style SGI Onyx performs about 50 % slower than the new Origin 2000, although the CPUs are identical. This is due to the new memory management architecture.

The SUN Ultra-Sparc 1 is plotted here really for reference to indicate its slow performance.

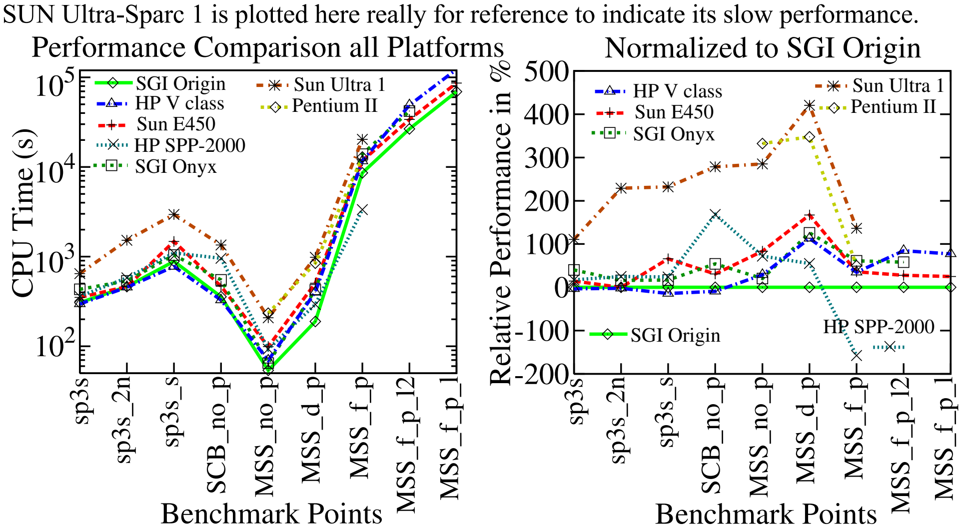

Fig. 7: Benchmarks for 7 different platforms. Besides the previously discussed three platforms 1) SGI Origin 200, 2) HP V class and 3) Sun E450, we test other platforms

available in the HPC group: 4) HP SPP-2000, 5) SGI Onyx, 6) Sun Ultra 1, and 7) Pentium II under LINUX. The HP SP-2000 and the SGI Onyx provide competitive performance. (b) shows the data of (a) normalized to the Origin 2000 data.

6 Summary and Conclusion

We have benchmarked the NEMO code on a variety of different machines to evaluate them as possible development platform for continued NEMO development. A variety of different physical models, which are going to be combined in future NEMO version were benchmarked. These benchmarks cover recursive operations on a large number of small matrices to operations on medium size full matrices. We also tested the compilation speed of the cc, f77 and f90 compiler on the machine on the 250,000 lines, 570 file NEMO code.

On all the benchmarks the SGI Origin 2000 has to be declared the winner in terms of raw performance . This is with the one caveat that parallelization for more than 2 CPUs could not be achieved for large memory hungry (285MB and 380 MB) benchmarks. The SGI compilers are very stable and do not require special attention for any one of the 570 source files. The SGI development environment is outstanding and stable (I am stating this from past, 1/2 year old experience). Origin 2000 processors are presently shipped running at 250 MHz compared to the 200 MHz benchmarks. A linear speed-up of 200/250=80% was verified on some of the benchmarks. This makes the SGI platform even more competitive to the HP and SUN.

The SUN shines in its almost perfect utilization of parallelization and in its compiler stability. None of the source files had to be treated special in any of the compiler options. However its overall speed performance lags the SGI by typically 20-100 %. SUN was kind enough to provide the time of several software and systems engineers to introduce their hardware and software. However even their experts had trouble using their newly reworked development environment (workshop). Workshop has many (well-hidden) options and I experienced several bugs/hang-up and crashes during my limited use. The compiler options to analyze the code for performance improvements (loop tools, memory tools) have not become clear in the benchmarking phase. Workshop appears to have the right tools needed for performance tuning, however SUN engineers point to a third party product to really do the line-by-line performance tuning.

The HP V Class running at 200 MHz lies typically lies between the SGI and SUN performance except for the compilation times, where it lags the two other platforms. Not only are the compilers slow, but they also crash on some of the F90 objects in the NEMO code, when maximum optimization is requested. This makes customizing of the makefile for a few objects necessary. We find that the third party F90 to F77 translator product from Pacific Sierra still outperforms the native HP F90 compiler on some of the benchmarks. The HP F90 compiler appears certainly not as stable as the SUN and SGI F90 compilers. The PA 8000 processors in the HP are now also shipped with a speed of 233 MHz. A performance enhancement of the ration 200/233=85% can be expected. HP offers a development environment called Softbench, which was not tested at this time. From personal 1/2 year old experience I can state that the Softbench environment is not competitive with the SGI environment. In the Softbench version used at Raytheon during the NEMO development no structure browsers, data browsers, memory checkers or automated performance tools were available. The HP V Class is out of the price range we considered. In the price range are the J and K Class machines with a maximum of 4 and 2 CPUs, respectively.

The direct competitor to the SUN and HP machines we tested is the SGU Origin 200, not the Origin 2000. The Origin 200 ships with 180 MHz processors and the benchmark time are expected to scale as 200/180=111%. The Origin 200 is limited to 4 CPUs, while the Origin 2000 is scalable beyond 4. Also the Origin 2000 is now shipping with 250MHz CPUs. I recommend the purchase of an Origin 2000.

7 Acknowledgements

I would like to acknowledge the help I received from the various computer vendors (SUN, HP, and SGI). In particular SUN and HP were kind enough to provide a dedicated machine for my benchmarks. Within JPL I got access to non HPC machines such as Neptune (SPP-2000), Yngvi (Origin 2000) and Castor and Pollux (Onyx) where the respective systems administrators helped significantly (Cris Windoffer, Edith Huang, Kathya Zamora, and Eric Danielson).

8 Appendix

8.1 Where is Compiler-Based Parallelization Useful?

For the three development platforms all benchmarks were executed for 1, 2, 3, and 4 CPUs. For the NEMO code in its present design we find that only problems which access the LAPACK libraries with larger matrix sizes benefit from the automatic parallelization. These problems are the full matrix operations in the benchmarks MSS_f_p, MSS_f_p_l2 and MSS_f_p_l. Some of the other scattering benchmarks MSS_d_p and SCB_n_p benefit slightly from the automatic parallelization. The bandstructure models do not benefit from parallelization at all. The detailed results for the three development platforms are given in the following Sections.

8.1.1 SUN Parallelization for all Benchmark Points

The execution times for 1, 2, 3, and 4 CPUs on the SUN E450 server are depicted in Fig. 8. Only the full matrix operations (MSS_f_p...) benefit significantly from compiler-based parallelization. The bandstructure-based benchmarks are hindered by parallel execution and the execution time rises.

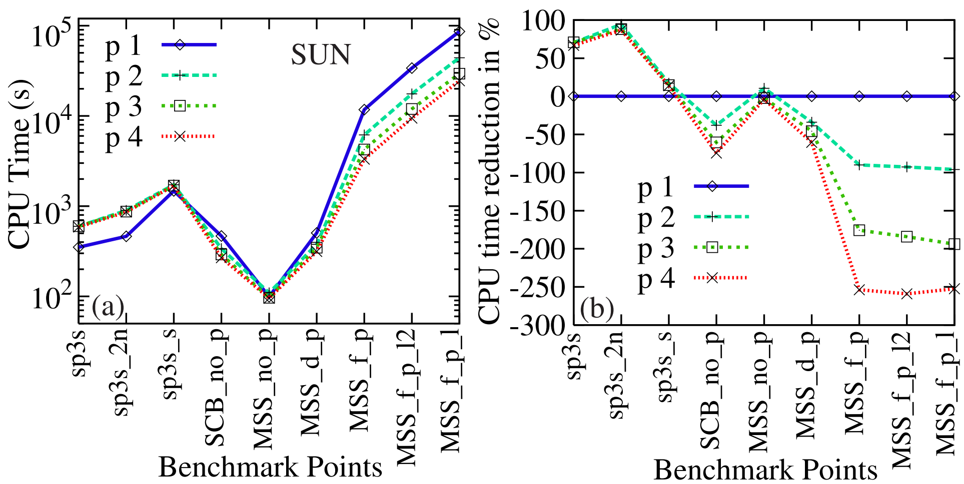

Fig. 8: Benchmarks executed on the SUN E450 server for 1, 2, 3, and 4 CPUs. (b) Contains the data of (a) normalized to the single CPU performance.

8.1.2 HP Parallelization for all Benchmark Points

The execution times for 1, 2, 3, and 4 CPUs on the HP V Class server are depicted in Fig. 9. The results are similar to the SUN E450 server results depicted in Fig. 8 however the HP executable does not suffer negatively from parallelization unlike the SUN.

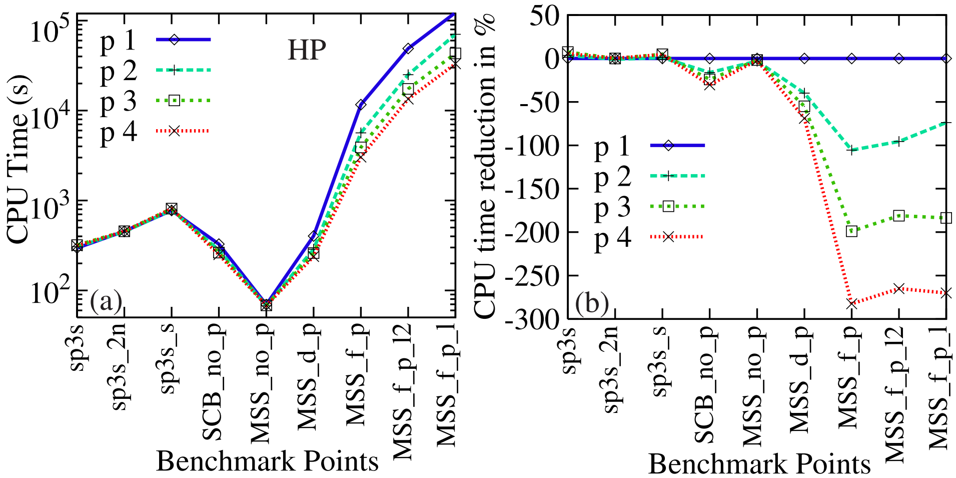

Fig. 9: Benchmarks executed on the HP V Class server for 1, 2, 3, and 4 CPUs. (b) contains the data of (a) normalized to the single CPU performance.

8.1.3 SGI Parallelization for all Benchmark Points

The execution times for 1, 2, 3, and 4 CPUs on the SGI Origin 2000 server are depicted in Fig. 10. On the SGI platform the compiler and runtime environment are smart enough to not even start separate threads for the "sp3s.." benchmark points. Only a single CPU is assigned to these benchmarks.

The analysis of the parallelization capabilities was severely handicapped on the available machine for the memory intensive benchmarks "MSS_f_p_l2" and "MSS_f_p_l". The execution of these benchmark points with more then 2 CPUs ended consistently with a segmentation fault. According to a SGI representative this problem can be fixed by setting some runtime limits. The parallelization efficiency is not as good as the ones achieved on the SUN and HP platforms.

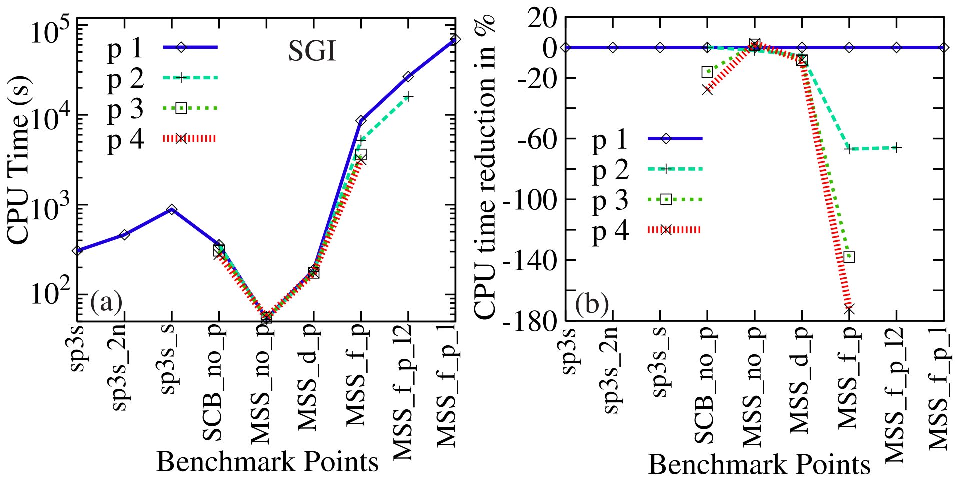

Fig. 10: Benchmarks executed on the SGI Origin 2000 server for 1, 2, 3, and 4 CPUs. (b) Contains the data of (a) normalized to the single CPU performance.

8.2 Parallelization with more than 4 CPU's

The two full matrix benchmarks MSS_f_p and MSS_f_p_l2 that we examined earlier for parallelization on the 4 CPU platforms in Fig. 4 and Fig. 5 are examined for larger scale parallelization on the SGI Origin 2000 and the HP SPP-2000 in Fig. 11. Fig. 11 (a) and (b) show the overall runtime of the two benchmarks as a function of CPUs on a log-log scale. Ideal is included in the figures for reference. Fig. 11

(c) and (d) show CPU time reduction factor normalized to the single CPU performance. This CPU time reduction factor scales ideally as the identity function. The data from Fig. 4 and Fig. 5 are included as a reference.

It is interesting to note that the HP SPP-2000 outperforms the Origin 2000 on the smaller memory requirement benchmark (MSS_f_p). As mentioned in Section 4.3 we did not get the Origin 2000 to run the MSS_f_p_l2 benchmark on more than 2 CPUs. Note, however that the SGI outperforms the HP SPP-2000 by at least a factor of 2 in that benchmark. In fact the SPP-2000 is the slowest one of the parallel machines on that benchmark (except for the old style SGI Onyx).

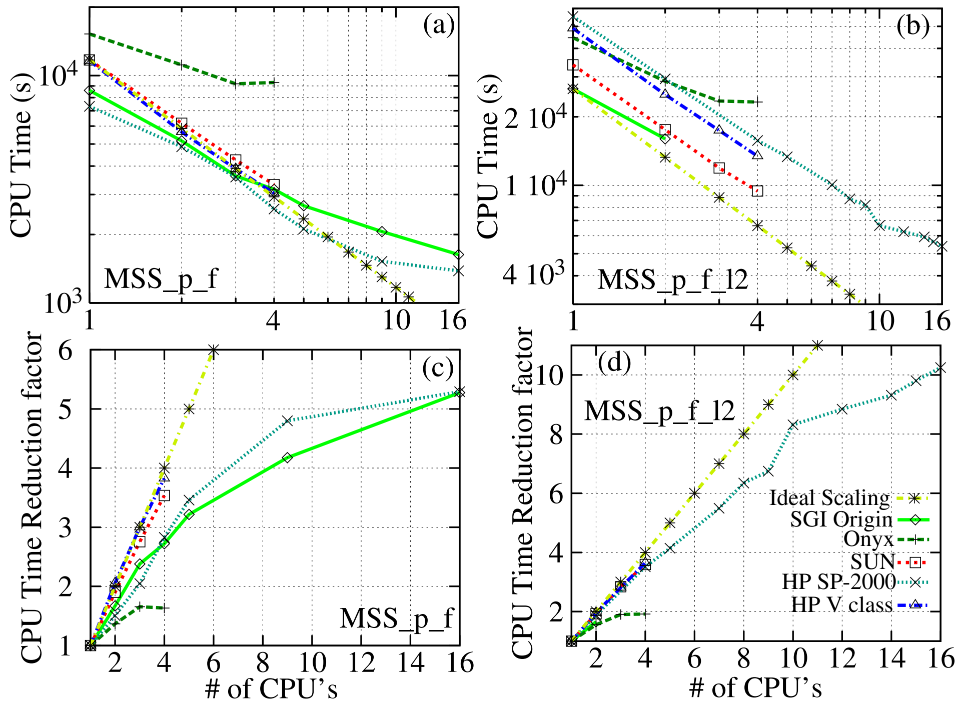

Fig. 11: Parallelization of the MSS_f_p ((a) and (c)) and MSS_f_p_l2 ((b) and (d)) benchmarks for 1-16 CPUs. (a) and (b) show the raw computation time as a function of CPUs on a log-log scale. Ideal scaling (100% efficiency) results in a straight line. (c) and (d) show the CPU time reduction factor normalized to single CPU performance. Ideal scaling is linear on this plot as well.

8.3 Compiler Dependence of the Individual Platforms

8.3.1 Development Platforms

8.3.1.1 SGI Origin 2000 (200 MHz)

Fig. 12 shows the performance of the Origin 2000 system in form of raw data in seconds and normalized to the code compiled natively on the machine using the SGI 7.2 compilers with automatic parallelization. Using the parallelized code on a single CPU provides a significant performance gain for 4 benchmark points and a slight loss in performance in 4 others. Native compilation with the latest compilers provides speedups of 20-135%. Especially the codes associated with the scattering simulation (larger matrices) appear to benefit from the new compilers.

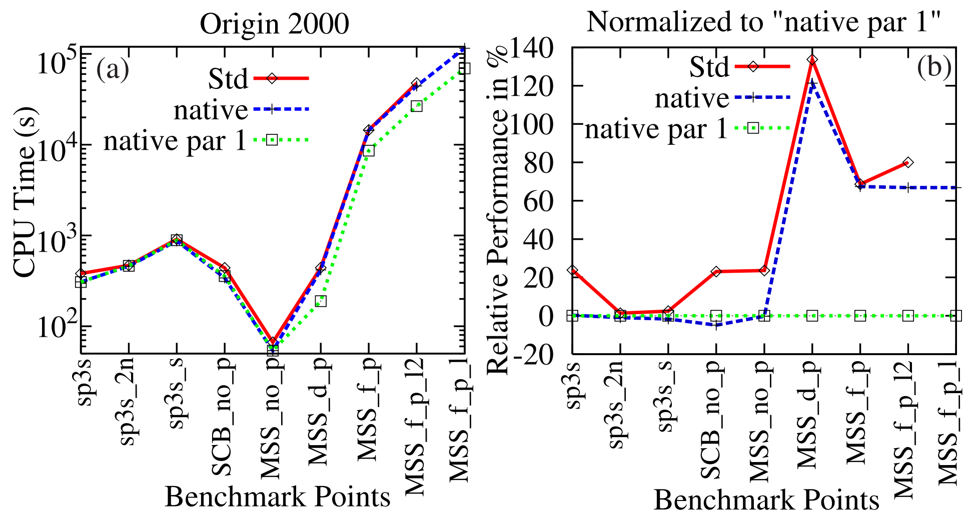

Fig. 12: Benchmark performance of the Origin 2000 system (200 MHz R10k) for three differently compiled codes: 1) Std: Raytheon code distribution, 2) native: SGI compilers version 7.2 compiled for single CPU. 3) native par 1: SGI compilers with automatic parallelization, run on a single CPU. The native parallel code runs fastest in most benchmark points and is used as normalization in (b).

Std	Compilers	SGI 6.1
	CFLAGS	-64 -woff 1048 -D_64BIT -O3 -OPT:fold_arith_limit=6000:const_copy_limit=12000 -fullwarn
	FFLAGS	-64 -O3 -OPT:alias=restrict:fast_complex=on special for sb_boundary_mi1_f.o due to compiler problems -64 -OPT:alias=restrict:fast_complex=o
	F90FLAGS	-64 -O3 -Ddimag_f90=dimag -Dcdabs=abs -Dcdsqrt=sqrt -Ddcmplx_f90=dcmplx -Ddconjg=conjg -OPT:alias=restrict:fast_complex=on:const_copy_limit=12000
	LDFLAGS	
	LIBS	-lcomplib.sgimath -lftn90 -lftn -lc -64
Native	Compilers	SGI 7.2
	CFLAGS	-64 -I$(INCL) -woff 1048 -D_64BIT -O3 -OPT:const_copy_limit=12000 -fullwarn -r10000 -Ofast=ip27 -IPA
	FFLAGS	-64 -O3 -OPT:alias=restrict:fast_complex=on -Ofast=ip25 -r10000 -Ofast=ip27 -IPA
	F90FLAGS	-64 -O3 -Ddimag_f90=dimag -Dcdabs=abs -Dcdsqrt=sqrt -Ddcmplx_f90=dcmplx -Ddconjg=conjg -OPT:alias=restrict:fast_complex=on:const_copy_limit=12000 -r10000 -Ofast=ip27 -IPA
	LDFLAGS	-IPA -Ofast=ip27 -r10000 -64
	LIBS	-B dynamic -lcomplib.sgimath -lfortran -lc -64 -lm
Native par	Compilers	SGI 7.2
	CFLAGS	-64 -I$(INCL) -woff 1048 -D_64BIT -O3 -OPT:const_copy_limit=12000 -fullwarn -r10000 -Ofast=ip27 -IPA -pca -mp
	FFLAGS	-64 -O3 -OPT:alias=restrict:fast_complex=on -Ofast=ip25 -r10000 -Ofast=ip27 -IPA -pfa -mp -mp_keep
	F90FLAGS	-64 -O3 -Ddimag_f90=dimag -Dcdabs=abs -Dcdsqrt=sqrt -Ddcmplx_f90=dcmplx -Ddconjg=conjg -OPT:alias=restrict:fast_complex=on:const_copy_limit=12000 -r10000 -Ofast=ip27 -IPA -pfa -mp
	LDFLAGS	-IPA -Ofast=ip27 -r10000 -64 -mp
	LIBS	-B dynamic -lcomplib.sgimath_mp -lfortran -lc -64 -lm
debug	Compilers	SGI 7.2
	CFLAGS	-64 -g
	FFLAGS	-64 -g
	F90FLAGS	-64 -g
	LDFLAGS	
	LIBS	-B dynamic -lcomplib.sgimath -lfortran -lc -64 -lm

Table 2: Compilers and compiler flags used on SGI Origin 2000 platform.

8.3.1.2 SUN E450 (300 MHz) Ultra Sparc 2

Fig. 13 shows the benchmark runs on the Enterprise 450 Sun Ultra Sparc 2 machine for three differently compiled versions of the NEMO code. The native code instrumented by automatic parallelization running on a single CPU performs best on most of the benchmark points. The relative performance shown in Fig. 13(b) shows that the improvements from the older compilers used in the Raytheon code to the newer compilers native to the Sparc Ultra 2 are rather sporadic. 2 benchmarks show large improvements of 90%

(sp3s_2n) and 70% (SCB_no_p), however only marginal improvements <20% on all other benchmarks. The sp3s and sp3s_s benchmark are numerically very similar to the sp3s_2n benchmark, but the codes are differing speed only by <10%.

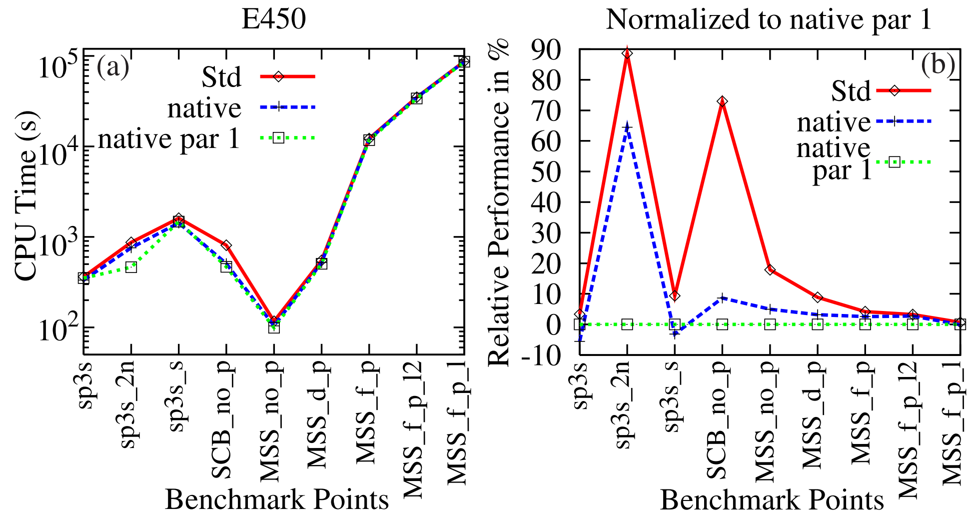

Fig. 13: Benchmark performance of the Sun E450 system (300 MHz Ultra Sparc) for three differently compiled codes: 1) Std: Raytheon code distribution, 2) native, Sun Workshop compilers compiled for a single CPU, 3) native par 1: Sun Workshop compilers with automatic parallelization. The native parallel code runs the fastest on most benchmark points and is used as normalization in (b).

The compilers, compiler options and linked libraries are shown in Table 3.

Std	Compilers	V3.X compiled on a Sparc 10
	CFLAGS	-fast -xO5
	FFLAGS	-fast -O5
	F90FLAGS	-fast -O4
	LDFLAGS	
	LIBS	-z muldefs -B static -lfast -llapack -lblas -lM77 -lF77 -lf90 -lm -lsunmath
Native	Compilers	SUNWspro/SC4.2
	CFLAGS	-fast -xO4 -xtarget=native -xsb -xautopar -Zlp
	FFLAGS	-c -fast -O5 -xtarget=native -xsb
	F90FLAGS	-fast -O3 -xtarget=native
	LDFLAGS	-fast -I
	LIBS	-z muldefs -L/opt/SUNWspro/SC4.2/lib -lfast -lM77 -lF77 -lf90 -lm -xlic_lib=sunperf
Native par	Compilers	SUNWspro/SC4.2
	CFLAGS	-fast -xO5 -xautopar
	FFLAGS	-fast -O5 -xtarget=native -xsb -autopar -parallel
	F90FLAGS	-fast -O3 -xtarget=native -xsb -autopar -parallel
	LDFLAGS	-fast -i -xautopar
	LIBS	-L/opt/SUNWspro/SC4.2/lib -lfast -lM77 -lF77 -lf90 -lm -xlic_lib=sunperf
debug	Compilers	SUNWspro/SC4.2
	CFLAGS	-g
	FFLAGS	-g -u
	F90FLAGS	-g
	LDFLAGS	-g
	LIBS	-z muldefs -L/opt/SUNWspro/SC4.2/lib -lfast -lM77 -lF77 -lf90 -lm -xlic_lib=sunperf

Table 3: Compilers and compiler flags used on SUN platforms.

8.3.1.3 HP V Class 80000 (200 MHz)

Fig. 14 shows the performance of a 200 MHz HP V class with PA 8000 processors running at 200 MHz. Thee differently compiled versions of the NEMO code are compared: 1) the standard Raytheon distribution (compiled on a HP 735), 2) a native compilation without parallelization and 3) a native compilation with parallelization turned on. The native compilation appears to have a very large effect for the scattering code, which uses iterative schemes and large matrix manipulation as see from the normalized graph in Fig. 14b.

The native (non-parallel) code does not converge on the SCB benchmark problem! The parallelized code generally outperforms the non-parallelized code on a single CPU. There is one interesting data point to note in the sp3s benchmark. The standard code outperforms the native code. This is attributed to the compilation of the standard code using the Pacific-Sierra F90 to F77 translator/compiler. The native HP F90 compiler is outperformed by the third party F90 compiler for small matrix operations. This can be seen more clearly in the other HP platform (HP/Exemplar SPP-2000) discussed in Section 8.3.1.1. The native HP F90 compiler still has some problems with 3 objects of the NEMO code as indicated in Table 4. The compiler crashes, if aggressive optimization is performed. This is an improvement to the compiler on the HP/Exemplar SPP-2000 where only a +O2 option can be used on these files, preventing a automatic parallelization (which demands +O3).

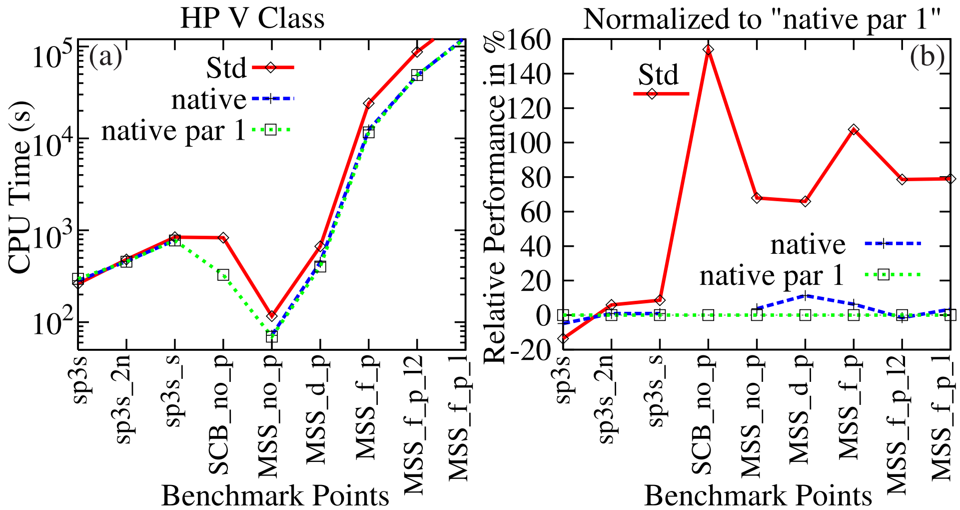

Fig. 14: Benchmark performance of a HP V class (PA8000 at 200 MHz) for three differently compiled codes: 1) Std: standard Raytheon distribution, 2) native: compiled on V class with optimized libraries, and 3) native par 1: compiled for multi CPU execution running on a single CPU. The performance is normalized to the parallel code in (b).

Std	Compilers	HP 735, OS 9.0.5 cc, f77, Pacific Sierra vast f90
	CFLAGS	-Aa +O3 +Oprocelim +Ofastaccess +Olimit +w1
	FFLAGS	+OPP +OP4 +O3
	F90FLAGS	+Obb1200 +OP4P +OPunroll -keep -
	LDFLAGS	+O3
	LIBS	-L/usr/local/lib -llapack -lblas -lvast90 -lcl -lisamstub -lvec
Native	Compilers	F77 Exemplar V1.2.2, F90 B.11.00..01, CC V1.2.2
	CFLAGS	-Aa +O3 +Oaggressive +Onolimit +Ofltacc +Odataprefetch +Olibcalls
	FFLAGS	+O3 +Oaggressive +Onolimit +Ofltacc +Odataprefetch +Olibcalls
	F90FLAGS	+O3 +Ovectorize +Oaggressive +Onolimit +Ofltacc +Odataprefetch +Olibcalls special for mb_RGF_Nband_f90.o cblockmatrix_f90.o mb_bc_micro_f90.o due to compiler core dump: +O3 +Ofltacc +Odataprefetch +Olibcalls
	LDFLAGS	+O3 -Wl,-a,archive_shared -Wl,+FPD
	LIBS	-Wl,-L/opt/mlib/lib/pa2.0 -lveclib -llapack -L/usr/local/lib -L/opt/fortran/lib -L/opt/fortran90/lib -lU77 -lF90 -lcl -lisamstub
Native par	Compilers	F77 Exemplar V1.2.2, F90 B.11.00..01, CC V1.2.2
	CFLAGS	+O3 +Oaggressive +Onolimit +Ofltacc +Odataprefetch +Olibcalls
	FFLAGS	+O3 +Oaggressive +Onolimit +Ofltacc +Oparallel +Oautopar +Odataprefetch +Olibcalls
	F90FLAGS	+O3 +Ovectorize +Oaggressive +Onolimit +Ofltacc +Oparallel +Oautopar +Odataprefetch +Olibcalls special for mb_RGF_Nband_f90.o cblockmatrix_f90.o mb_bc_micro_f90.o due to compiler core dump: +O3 +Ofltacc +Oparallel +Oautopar +Odataprefetch +Olibcalls
	LDFLAGS	+O3 +Oparallel -Wl,-a,archive_shared -Wl,+FPD
	LIBS	-Wl,-L/opt/mlib/lib/pa2.0parallel -lveclib -llapack -L/usr/local/lib -L/opt/fortran/lib -L/opt/fortran90/lib -lU77 -lF90 -lcl -lcps -lpthread -lisamstub
debug	Compilers	F77 Exemplar V1.2.2, F90 B.11.00..01, CC V1.2.2
	CFLAGS	-g -z
	FFLAGS	-g -G -u
	F90FLAGS	-u -g -G
	LDFLAGS	-g
	LIBS	-Wl,-L/opt/mlib/lib/pa2.0 -lveclib -llapack -L/usr/local/lib -L/opt/fortran/lib -L/opt/fortran90/lib -lU77 -lF90 -lcl -lisamstub

Table 4: Compilers and compiler flags used on HP V Class platforms. The native f90 compiler crashes on the full optimization of three files and the optimization level needed to be reduced.

8.3.2 Alternate Platforms

8.3.2.1 SGI Onyx (200 MHz)

For reference the NEMO code was also recompiled for single and multiple CPU runs on an old style Onyx SGI platform. This machine has been replaced by the ONYX2 system. The machine was not yet upgraded to the new 7.2 compilers and the old 6.2 compilers were used. A code cross-compiled for IP25 was transferred from the Origin 2000 system and slightly faster results were obtained on some benchmarks. However on 3 out of 8 benchmarks a core dump occurred when the 7.2 compiled code was executed. The standard NEMO code was compiled at Raytheon on a Power Challenge using SGI 6.1 compilers. Only a small improvement in performance using slightly different compiler options can be seen on this platform.

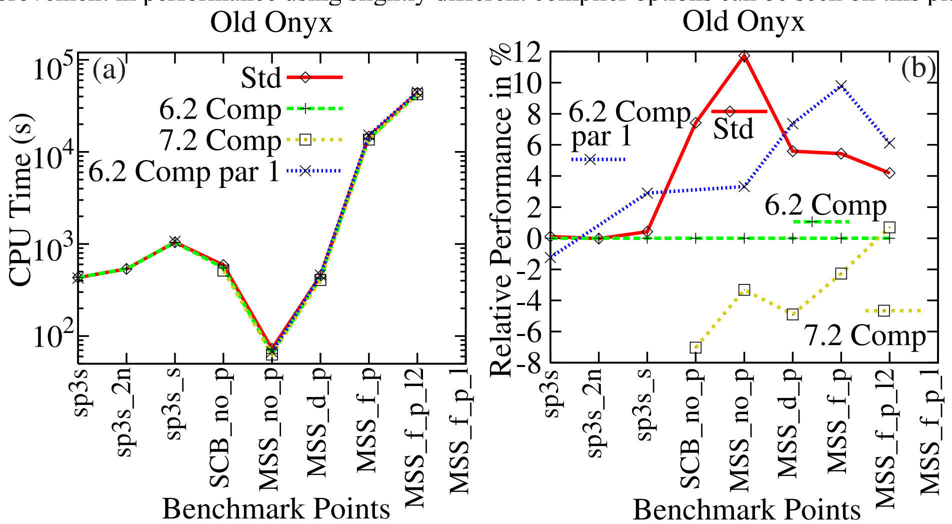

Fig. 15: Benchmark performance of a old Onyx (R10k at 200 MHz) SGI system for four differently compiled codes: 1) Std: Raytheon code distribution, 2) native SGI 6.2 Compiler, 3) native SGI 6.2 Compiler parallelized, and 4) native SGI 7.2 Compiler (cross compiled from a Origin 2000). The native unparallelized code runs the fastest (7.2 code fails on some benchmarks) and is used for a normalization in (b).

Std	Compilers	SGI 6.1
	CFLAGS	-64 -woff 1048 -D_64BIT -O3 -OPT:fold_arith_limit=6000:const_copy_limit=12000 -fullwarn
	FFLAGS	-64 -O3 -OPT:alias=restrict:fast_complex=on special for sb_boundary_mi1_f.o due to compiler problems -64 -OPT:alias=restrict:fast_complex=o
	F90FLAGS	-64 -O3 -Ddimag_f90=dimag –Dcdabs=abs -Dcdsqrt=sqrt -Ddcmplx_f90=dcmplx -Ddconjg=conjg -OPT:alias=restrict:fast_complex=on:const_copy_limit=12000
	LDFLAGS	
	LIBS	-lcomplib.sgimath -lftn90 -lftn -lc -64
Native	Compilers	SGI 6.2
	CFLAGS	-c -n32 -mips4 -I$(INCL) -woff 1048 -O3 -OPT:fold_arith_limit=6000:const_copy_limit=12000 -fullwarn -r10000 -Ofast=ip25 -IPA
	FFLAGS	-c -n32 -mips4 -O3 -OPT:alias=restrict:fast_complex=on -Ofast=ip25 -r10000
	F90FLAGS	-c -n32 -mips4 -O3 -Ddimag_f90=dimag -Dcdabs=abs -Dcdsqrt=sqrt -Ddcmplx_f90=dcmplx -Ddconjg=conjg -OPT:alias=restrict:fast_complex=on:const_copy_limit=12000 -r10000
	LDFLAGS	-IPA -Ofast=ip25 -r10000 -n32 -mips4
	LIBS	-B dynamic -lcomplib.sgimath -lftn90 -lftn -lc -lm
Native par	Compilers	SGI 6.2
	CFLAGS	-c -n32 -mips4 -I$(INCL) -woff 1048 -woff 1185 -O3 -OPT:fold_arith_limit=6000:const_copy_limit=12000 -r10000 -Ofast=ip25 -xansi -pca -mp special for ceigensyst.o, cnonsymeigsys.o, NemoOut_BX.o, and egrid.o due to compiler error or runtime crash: -c -n32 -mips4 -I$(INCL) -woff 1048 -woff 1185 -O3 -OPT:fold_arith_limit=6000:const_copy_limit=12000 -r10000 -Ofast=ip25
	FFLAGS	-c -n32 -O3 -OPT:alias=restrict:fast_complex=on -Ofast=ip25 -r10000 -pfa -mp -mp_keep special for cmatrix_f.o sb_scatter_GR0_f.o, and sb_is_GL0_f.o due to compiler errors (no parallelization) -c -n32 -mips4 -O -Ofast=ip25 -r10000
	F90FLAGS	-c -n32 -mips4 -Ddimag_f90=dimag -Dcdabs=abs -Dcdsqrt=sqrt -Ddcmplx_f90=dcmplx -Ddconjg=conjg -pfa -mp -O3 -OPT:alias=restrict:fast_complex=on:const_copy_limit=12162 -r10000
	LDFLAGS	-IPA -n32 -mips4
	LIBS	-B dynamic -lcomplib.sgimath_mp -lftn90 -lftn -lc -n32 -mp -lm

Table 5: Compilers and compiler flags used on SGI Onyx platform. The parallel options had to be customized for four C and three f77 files. This problem does not occur with the newer 7.2 compilers anymore.

8.3.2.2 HP/Convex SPP-2000 (180 MHz)

The HP SPP-2000 is a massively parallel supercomputer with 256 nodes. The individual CPU's are PA 8000 running at 180 MHz. The F90 code in NEMO was originally developed using a Pacific-Sierra third party F90 to F77 translator (Vast), since HP did not provide a native F90 compiler. HP entered the F90 development late compared to other high performance computer vendors and appears to suffer from this still. For several benchmark points the vast compiler still generates the faster code compared to the native HP compiler. Furthermore the native HP f90 compiler crashes on three files when the +O3 optimization is turned on. This prevents the automatic parallelization of these modules. Surprisingly the standard code (developed on PA 1.1 for HP 735) still runs faster than the native code on small complex matrix multiplication (sp3s benchmark). With this platform it is noticeable that for different benchmarks different compilers generate faster codes. There is no "best" compiler for all the benchmarks. For normalization we use the native compilation on the SPP-2000 with the Vast F90 compiler.

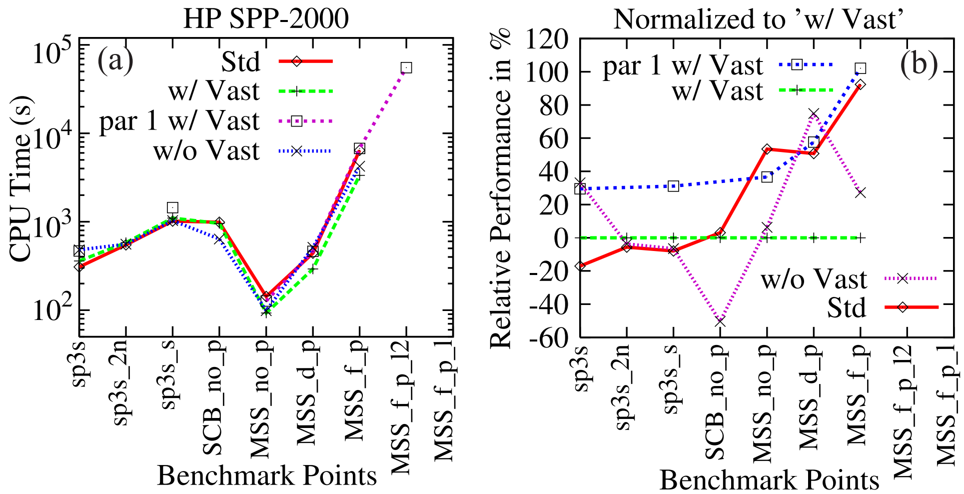

Fig. 16: Benchmark performance of the HP/Convex SPP-2000 massively parallel machine for single CPU performance. Several differently compiled codes are tested: 1) Std: Raytheon code distribution, 2) native cc, f77 and vast f90, 3) same as previous, but instrumented for parallel computation, and 4) native cc, f77 and f90 without parallelization.

The benchmarking for the performance of a few CPUs is somewhat hindered by problems in the queuing system on the SPP-2000. Single CPU jobs are frequently oversubscribed with other processes, which results in reduced performance.

Std	Compilers	HP 735, OS 9.0.5 cc, f77, Pacific Sierra vast f90
	CFLAGS	-Aa +O3 +Oprocelim +Ofastaccess +Olimit +w1
	FFLAGS	+OPP +OP4 +O3
	F90FLAGS	+Obb1200 +OP4P +OPunroll -keep -
	LDFLAGS	+O3
	LIBS	-L/usr/local/lib -llapack -lblas -lvast90 -lcl -lisamstub -lvec
w/ Vast	Compilers	Native cc and f77, Pacific Sierra vast f90
	CFLAGS	+O3 +Oaggressive +Onolimit +Ofltacc +Oinfo
	FFLAGS	+O3 +Oaggressive +Onolimit +Ofltacc +Oinfo +Oreport=all +Ovectorize
	F90FLAGS	+O3 +Oaggressive +Onolimit +Ofltacc +Oinfo +Oreport=all +Ovectorize -keep -w
	LDFLAGS	+O3
	LIBS	-Wl,-L/opt/mlib/lib/pa2.0parallel -lveclib -llapack -L/usr/local/lib -L/opt/fortran/lib -lU77 -lfsys -lcl -lcps /opt/vast90/lib/libvast90.a
w/ vast par	Compilers	Native cc and f77, Pacific Sierra vast f90
	CFLAGS	+O3 +Oaggressive +Onolimit +Ofltacc +Oinfo +Oparallel
	FFLAGS	+O3 +Oaggressive +Onolimit +Ofltacc +Oinfo +Oreport=all +Ovectorize +Oparallel
	F90FLAGS	+O3 +Oaggressive +Onolimit +Ofltacc +Oinfo +Oreport=all +Ovectorize +Oparallel -keep -w
	LDFLAGS	+O3 +Oparallel
	LIBS	-Wl,-L/opt/mlib/lib/pa2.0parallel -lveclib -llapack -L/usr/local/lib -L/opt/fortran/lib -lU77 -lfsys -lcl -lcps /opt/vast90/lib/libvast90.a
W/o vast	Compilers	Native f77, f90, and cc
	CFLAGS	+Oall +Oaggressive +Onolimit +Ofltacc
	FFLAGS	+Oall +Oaggressive +Onolimit +Ofltacc +Opunroll special for mb_RGF_Nband_f90.o, cblockmatrix_f90.o, mb_bc_micro_f90.o due to compiler crash with optim. +O2 +Ovectorize +Oreport=all
	F90FLAGS	+Oall +Ovectorize +Oaggressive +Onolimit +Ofltacc +OPunroll
	LDFLAGS	+O3
	LIBS	-Wl,-L/opt/mlib/lib/pa2.0parallel -lveclib -llapack -L/usr/local/lib -L/opt/fortran/lib -L/opt/fortran90/lib -lU77 -lF90 -lfsys -lcl -lcps

Table 6: Compilers and compiler flags used on HP/Convex SPP-2000 platform. The native f90 compiler crashes on the optimization of 3 files. The optimization needs to be reduced to +O2 which prevents automatic parallelization.