Science & Technology Facilities Council

Technical Report
DL-TR-2007-005

A Study of the Performance of LAPACK Symmetric Matrix Diagonalisers on Multi-core Architectures

IN Kozin, MJ Deegan

November 2007

Chadwick Library
Science and Technology Facilities Council
Daresbury Laboratory
Daresbury Science and Innovation Campus
Warrington
WA4 4AD

Tel: +44(0)1925 603397
Fax: +44(0)1925 603779
email: library@dl.ac.uk

Science and Technology Facilities Council reports are available online at: http://epubs.cclrc.ac.uk/

ISSN 1362-0207

A Study of the Performance of LAPACK Symmetric Matrix Diagonalisers on Multi-core Architectures

Igor N. Kozin and Miles J. Deegan
Computational Science and Engineering Department, STFC Daresbury Laboratory, Daresbury, Warrington, Cheshire, WA4 4AD, UK

Abstract

The threaded performance of real symmetric dense matrix diagonalisers has been thoroughly investigated. Two problems have been identified as preventing efficient performance and good scaling. First, the library must be fully threaded as opposed to a partially threaded library. Second, there is a more fundamental, algorithmic problem which results in memory bandwidth bottleneck during the reduction of a matrix to tri-diagonal form. The tri-diagonalisation step starts dominating from certain matrix sizes and as a result using multi-cores in this situation becomes inefficient. Thus diagonalisation algorithms more suitable for multi-cores are required.

21/09/2007

Introduction

It is widely acknowledged that with the advent of multi-core processors the “free lunch” provided by ever increasing CPU clock frequencies is over, and that the HPC community (and indeed IT industry as a whole) needs to find alternative routes to increased application performance, namely ways of identifying and efficiently implementing parallelism.

The best solution is clearly to redesign and engineer algorithms and codes with multi/many-core architectures in mind from the outset. This approach however may be uneconomic or impractical. There are potentially easier routes to performance gains through parallelism. One route is to use auto-parallelising compilers, but unfortunately this ‘holy grail’ has yet to be delivered, and speed-ups obtained through this approach tend to be disappointing . For codes that rely on numerical libraries – for example FFTW[1] and LAPACK[2] (a cornerstone of HPC providing the basis for selecting the Top 500 supercomputers[3]) – there is potential for performance gains if efficiently threaded versions of these libraries are available.

LAPACK is an upper layer library which sits on top of another library – BLAS (Basic Linear Algebra Subprograms). The two libraries are separate but sometimes when LAPACK is referred to it implicitly includes BLAS. Hardware vendors will typically offer a BLAS and LAPACK implementation optimised for the platforms they offer. Ultimately all the LAPACK implementations are derived from the netlib.org source. Among the most popular ones are MKL (Intel)[4] and ACML (AMD)[5]. HP offers LAPACK functionality for its legacy architectures such as CXML for Alpha and MLIB for PA-RISC and Itanium. Subroutines

[1] Fastest Fourier Transform in the West (FFTW) http://www.fftw.org/
[2] Linear Algebra PACKage (LAPACK) http://www.netlib.org/lapack/
[3] Top 500 http://www.top500.org
[4] Intel Math Kernel Library (MKL) http://www.intel.com/cd/software/products/asmo-na/eng/307757.htm
[5] AMD Core Math Library (ACML) http://developer.amd.com/acml.jsp

from LAPACK have also found their way into IBM's Engineering Scientific Subroutine Library (ESSL). All these libraries come with very efficient BLAS implementations. Nevertheless there are very competitive non-commercial alternatives such as the self-tuning ATLAS suite[6] and the BLAS being developed by Goto[7] which is free for academic use. Thus an interesting alternative can be achieved by supplementing ATLAS or the GotoBLAS by the LAPACK built from source which has recently been updated to version 3.1.1.

Multi-threading BLAS is relatively straightforward and many implementations support multi-threading. Thus this work explores the route of "parallelisation through parallel libraries" and more specifically threaded LAPACK libraries. Furthermore, we limit ourselves to a subset of LAPACK functionality which is extremely important in many applied problems, namely real symmetric diagonalisers. The need for eigenvectors or/and eigenvalues of a matrix often appears in many fields ranging from physics and chemistry through to engineering and even computational finance. Efficient diagonalisers are important because a large portion of an application's execution time may be spent in such routines. Although diagonalisers are floating point intensive[8] matrix operations can be made "cache friendly", and therefore memory bandwidth bottlenecks may be partially circumvented.

In this paper we assess the performance of LAPACK libraries on various commodity architectures with an emphasis on multi-core performance and threaded scaling. Only threaded libraries will be analysed but we plan to benchmark MPI-based diagonalisers in future work.

The benchmark

The time it takes to diagonalise a matrix is obviously a function of its size but also to a lesser degree of its type. Benchmarking many different types of matrices would however be rather time consuming. The matrix we use appears in an application called WAVR4[9] which computes spectra of tetra-atomic molecules. We hope that this example is reasonably typical for a range of applications in the physical sciences. The matrix was generated for one particular problem and saved as an ASCII file for portability. It is real and symmetric, has no degenerate eigenvalues and its spectrum is well spread out. A graphical view of the matrix is given in Figure 1. The full matrix size is 7075 but we also used half of it (3575, lower left corner of Figure 1). In double precision these matrices take about 400 MB and 100 MB respectively, and therefore will not be cache-resident. Larger matrices could easily be generated but the run time grows as N^3 , and apart from slight differences in scaling, we did not find enough justification for using larger matrix sizes in this study.

[6] Automatically Tuned Linear Algebra Software (ATLAS) http://math-atlas.sourceforge.net/
[7] GotoBLAS http://www.tacc.utexas.edu/resources/software/
[8] Typically double precision is required although there have been recently some works done on incorporating single precision calculations in the double precision context (see for example, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, P. Luszczek, and S. Tomov, "Exploiting Mixed Precision Floating Point Hardware in Scientific Computations";
http://www.netlib.org/utk/people/JackDongarra/PAPERS/par_comp_iter_ref.pdf).
[9] I.N. Kozin, M.M. Law, J. Tennyson, J.M. Hutson, "New vibration-rotation code for tetraatomic molecules exhibiting wide-amplitude motion: WAVR4", Comp. Phys. Comm. 163 (2): 117-131 (2004).

LAPACK was originally implemented in Fortran but its success has led to implementations in other high level languages: C, C++, Fortran 95 and Java. We decided to implement our benchmark in Fortran 95 not only because this is in keeping with the original, but also

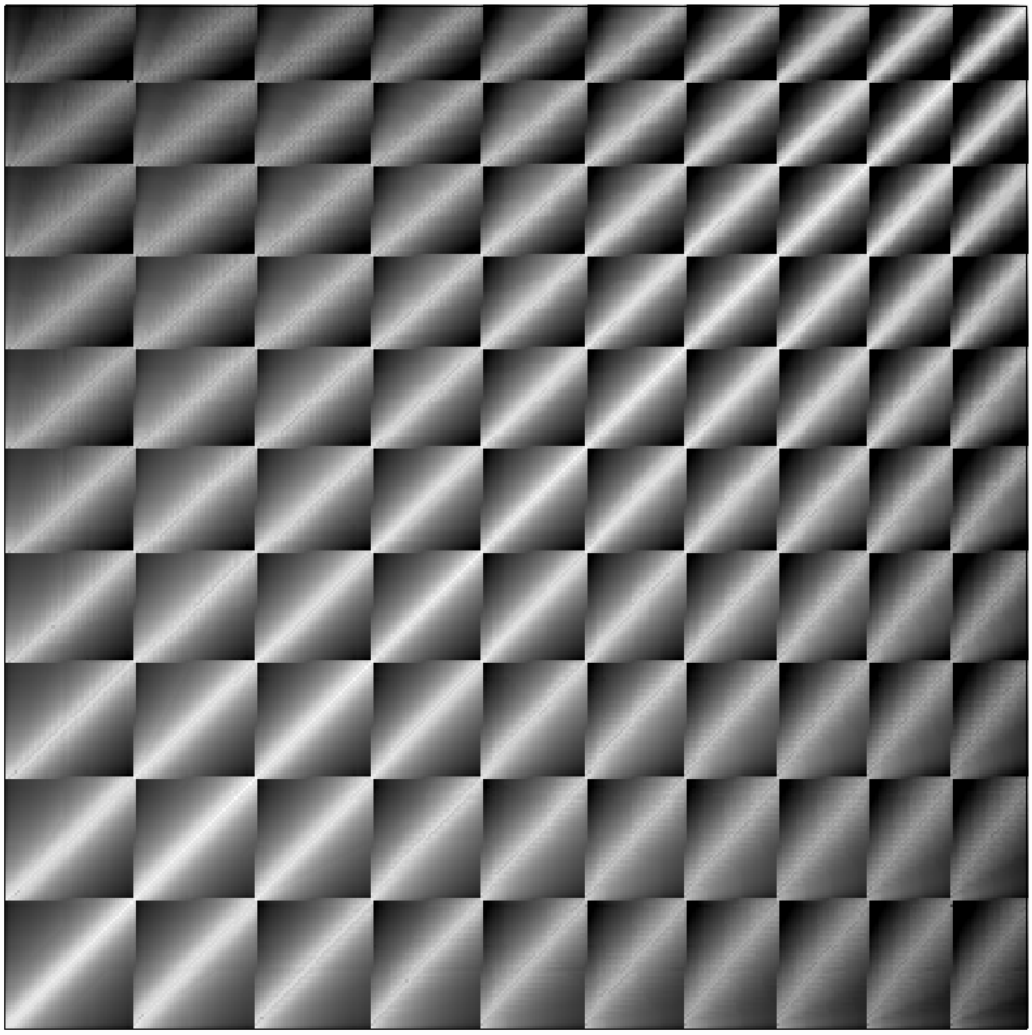

Figure 1: Graphical view of the matrix used in the tests: pixels from white to black represent aggregated matrix elements on the log scale with white being the largest.

because Fortran is still very widely and this situation is unlikely to change. The source is pure Fortran 95 (e.g. the wall-clock time is implemented using Fortran 95 DATE_AND_TIME) and is available from the Distributed Computing web site[10]. One further advantage in using Fortran 95 is to be able to use LAPACK95 interfaces and thus hide slight differences among LAPACK implementations. Through the command line options it is possible to vary the algorithms and stipulate whether or not eigenvectors are required. The number of threads is controlled via the environment variable OMP_NUM_THREADS.

The benchmark calls the four different diagonalisation algorithms implemented in LAPACK subroutines: DSYEV, DSYEVD, DSYEVR and DSYEVX. DSYEV is the basic diagonaliser

[10] Dbench benchmark can be downloaded from Distributed Computing Group website: http://www.cse.scitech.ac.uk/disco

which is used when all eigenvalues are needed. DSYEVD uses a divide and conquer algorithm to compute eigenvectors. DSYEVX and DSYEVR are more advanced and allow the computation of a selection of the eigenspectrum. DSYEVR is the most recent addition to LAPACK and employs Relatively Robust Representations. However, all these algorithms regardless of whether eigenvectors are required make the reduction of the matrix to tri-diagonal form their first step. The difference appears only later when the tri-diagonal matrix is being diagonalised.

Intel quad-core performance

Our main benchmarking system was a two socket Intel "Clovertown" server. Here is a summary of its technical details:

- CPU: 2x Xeon X5355 2.66 GHz
- Chipset: Blackford
- System bus: 1333 MHz
- Memory: 4x1GB 533 MHz FB-DIMMs
- Motherboard: Supermicro X7DB8+ rev 2.01
- OS: RHEL 4 update 4
- Intel compilers 10.0 (Build 20070613)

On paper this is a powerful system with a theoretical peak performance of 85 GFLOPS (8 cores capable of 4 fused add-multiply at 2.67 GHz). Indeed it can perform matrix-matrix multiplication using DGEMM at 68 GFLOPS (80% efficiency).

First, in order to assess the performance of matrix diagonalisers we obtained the reference serial performance of the latest LAPACK (3.1.1) built with Intel compilers. Then we compared its performance to the performance of other libraries[11] and the results are presented in Figure 2. We observe that Intel's MKL always gives the best performance as one might expect for an Intel library on an Intel platform. But other libraries are usually not too far behind. However Intel's DSYEV has managed to maintain a performance lead, particularly with the latest release of MKL (version 9.1.021).

Table 1: Time consumption breakdown (%) for different LAPACK algorithms within LAPACK built with Intel compilers.

	no vectors				vectors			
	dsyev	dsyevd	dsyevr	dsyevx	dsyev	dsyevd	dsyevr	dsyevx
dsymv	64	63	63	63	10	16	63	63
dsyr2k	26	27	27	27	4	7	27	27
dgemm					20	73		
dlasr					64			

One simple method of parallelising LAPACK is to use a threaded BLAS library. Fortunately one such implementation, GotoBLAS, is freely available for academic use. This provided us

[11] We have also built and used ATLAS 3.7.37 (currently the latest version). It recognised the hardware and the build was successful (we used GNU 4.2.1 compilers). However the performance was disappointing. Since version 3.7 is still in development, and therefore cannot be recommended for production usage, we decided not to include the ATLAS library in the benchmarking at this time. Furthermore, its matrix-vector operations are not threaded (read further on the relevance of this). We shall revisit ATLAS performance in a future paper.

with a reference for threaded performance. Although GotoBLAS substantially improves serial performance (we used version 1.16), the scaling was not very good. The reasons for this are the algorithms used in the LAPACK diagonalisers and the threaded implementation within GotoBLAS. In Table 1 below we present a profile of the main time consumers for all diagonalisers in question as implemented by LAPACK 3.1.1 (serial runs, matrix size 3575).

We observe that most of the tests are dominated by matrix-vector multiplications (DSYMV) and only some by matrix-matrix multiplications (DLASR, DGEMM). It turns out that GotoBLAS threading accelerates only DSYR2K and DGEMM (both are matrix-matrix multiplications). DLASR is part of LAPACK and therefore is not affected. Once GotoBLAS is used with LAPACK the profiles change a little but remain essentially the same. The profiles obtained for GotoBLAS 1.16 are presented in Table 2.

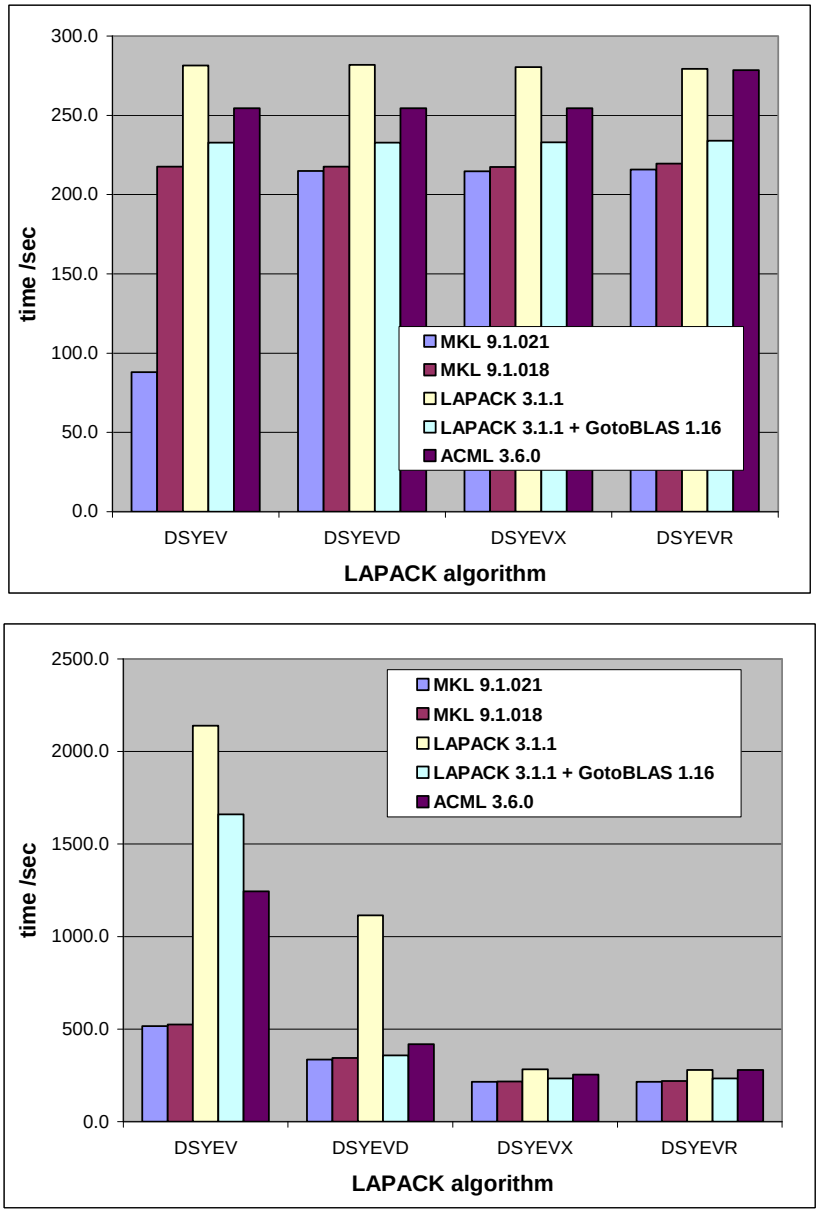

Figure 2: Serial performance of symmetric matrix diagonalisers on Intel Clovertown 2.66 GHz. Top: only eigenvalues are computed. Bottom: both eigenvalues and eigenvectors are computed. The matrix size is 7075.

Table 2: Time consumption breakdown (%) for different LAPACK algorithms in LAPACK built with Intel compilers and supplemented by GotoBLAS.

	no vectors				vectors			
	dsyev	dsyevd	dsyevr	dsyevx	dsyev	dsyevd	dsyevr	dsyevx
dgemv_t	61	62	62	60	9	39	62	62
dgemv_n	16	16	15	18	3	11	16	16
dgemm_kernel	11	12	12	11	5	34	12	11
dlasr					80			

Thus it should come as no surprise that LAPACK+GotoBLAS scaling is generally rather poor. Figure 3 presents the scaling of selected tests and all the data are available in the Appendix. Indeed DSYEVD demonstrates the best scaling achieving 40% improvement because DGEMM is accelerated by threading. Other tests show only around 10% speedup regardless of the number of threads. However even then the usefulness of DSYEVD is doubtful since in absolute terms other algorithms like DSYEVR and DSYEVX are quicker. It should be noted that as of version 1.18 GotoBLAS provides threaded DGEMV but the behaviour of DSYMV is not affected.

Figure 3 also summarises the scaling of MKL and ACML libraries. It shows threaded performance of MKL and LAPACK+GotoBLAS is reasonably close for all the tests except DSYEV. Although MKL performs better the scaling is little different. In contrast ACML

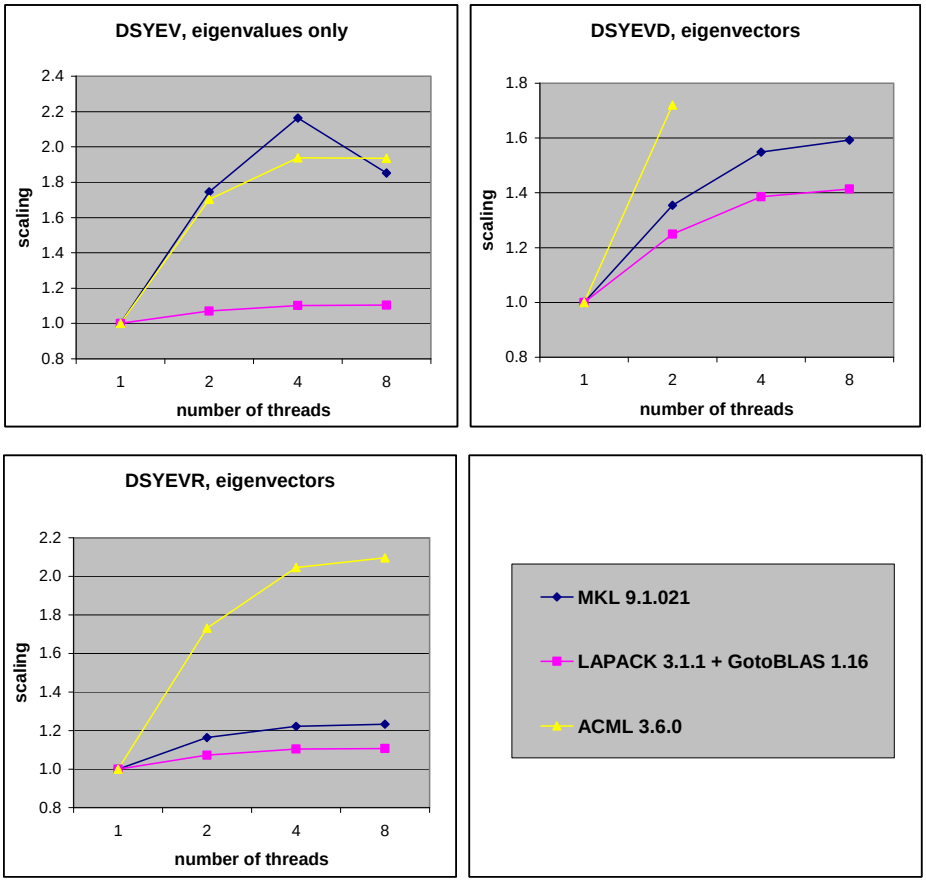

Figure 3: Scaling of different LAPACK libraries on Intel Clovertown 2.66 GHz. The matrix size is 7075.

presents a different pattern: often being slower on a single thread it scales better and as a result wins in most of the tests. Whether this is an indication of some algorithmic changes on the LAPACK level or not, we are not sure. Ultimately all the tested libraries struggle to achieve scaling better than a factor of two and this is clearly an algorithmic issue. As the profiles above demonstrate, all the subroutines heavily use matrix-vector products, the exceptions being DSYEV and DSYEVD when eigenvectors are needed. However this operation is limited: even if matrix-vector operations were threaded the performance would be throttled by the available aggregate memory bandwidth[12]. If we take 8GB/s as a rough estimate for practically achievable memory bandwidth, the maximum performance we can get on a matrix-vector product is only 2 GFLOPS (one double precision number is being fetched from memory roughly every nanosecond and used in two floating point operations – multiplication and addition). Compared to the peak of 85 GFLOPS this is only 2% efficiency. The measured floating point efficiency is in line with this number and was measured at around 3-4%. The higher average numbers are possible because there are other than tri-diagonalisation steps. In a spectacular fit, MKL's DSYEV (no eigenvectors) achieved 7% processor efficiency on one thread and 14% on eight. MKL's bus efficiency changes very little from one thread (typically at around 55%) to eight (around 60%). In contrast, ACML nearly doubles the bus efficiency from 44% on one thread to 78% on eight, which probably goes some way towards explaining the scalability.

As mentioned above, all the algorithms do reduction to tri-diagonal form as a first step and profiling shows that it takes up 90+% of the time. Indeed the reduction using DSYTRD

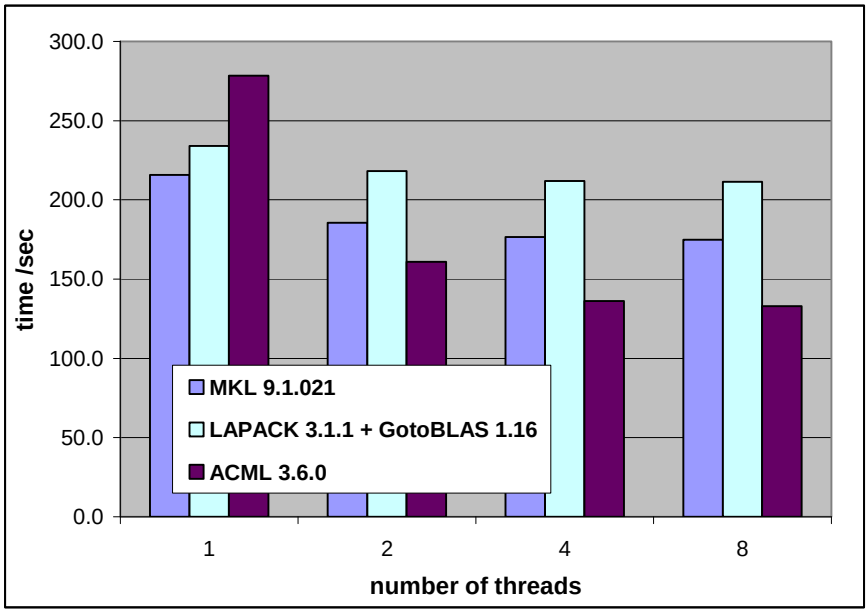

Figure 4: Threaded performance of DSYEVR from MKL, ACML and LAPACK+GotoBLAS on Intel Clovertown 2.66 GHz. Both eigenvalues and eigenvectors are computed. The matrix size is 7075.

requires $4/3N^3+O(N^2)$ floating point operations, whereas computing eigenvalues of a tri-diagonal matrix requires only $O(N^2)$ [13]. Our matrix size is sufficiently big that tri-diagonalisation dominates. This is the reason why performance of some tests is very close

[12] Our tests showed that matrix-vector operations provided by MKL and ACML are not threaded.

[13] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, "LAPACK Users' Guide", 3rd Edition 1999.

and why some of the profiles are so similar. In fact, the profiles of DSYEV and DSYEVD in Tables 1 and 2 differ so much from others when eigenvectors are requested because the subsequent steps are too slow and not because the tri-diagonalisation takes longer.

An examination of LAPACK libraries on other platforms

It is instructive to compare the scaling of threaded LAPACK offered by other libraries which are available on other hardware platforms. The purpose of this section is not so much the absolute performance comparison as the scaling with the number of threads. We have run the same benchmark on a number of systems:

- Supermicro X7DB8+ rev 2.01, Intel Clovertown 2.66 GHz, 4 GB RAM, MKL 9.1.021, ACML 3.6.0;
- Supermicro X7DB8+ rev 2.01, Intel Woodcrest 3.0 GHz, 4 GB RAM, MKL 9.1.021, ACML 3.6.0;
- SunFire Server V890: 8 UltraSparc IV (dual core) processors, 64 GB RAM, Sun Performance Libraries;
- IBM eServer 575[14]: 8 Power5 1.5 GHz (dual core), 32 GB RAM, ESSL;
- HP ProLiant DL145 Server: Opteron 280 2.4 GHz, 8 GB RAM, ACML 3.6.0;
- SGI Altix: Itanium 1.6 GHz, 96 GB RAM, MKL 9.1.021.

All the data are reported in the Appendix but a selection showing performance and scaling of DSYEV (all eigenvalues only) and DSYEVR (1% of eigenvectors) is presented in Figures 5–8. The results are surprising. First of all, ACML on AMD Opteron scales very poorly. Given that ACML performed so well on Clovertown this requires further investigation. Secondly, both the Sun and the IBM server demonstrate very good scaling although they are quite different in term of absolute performance. Their scaling is very close up to four threads after which the Sun server performance tails off. Good scaling is due in part to the availability of efficiently threaded libraries[15]. Finally, while the Intel machines dominate in single thread performance they get overtaken by the IBM server for multiple threads.

At this stage we will not speculate on how performance will be affected when Intel makes the transition to its CSI/QuickPath interconnect as a replacement for its current FSB technology and whether the memory latency and bandwidth of CSI (and for that matter Hypertransport) will be sufficient for codes such as these once we are operating in the many-core regime (16-32+ cores). It is likely however that the byte-per-flop ratio will continue to worsen for the foreseeable future and new programming strategies and models (with hardware support) are going to be required in order to fully exploit such architectures.

[14] One HPCx compute node (LPAR) has been used. One Power5 chip has 2 cores with 32KB data and 64KB instruction L1 cache each, 1.9 MB L2 cache is shared. Each chip is packaged together with 36 MB L3 cache (shared) into a dual-core module (DCM), i.e. 18 MB L3 per DCM. Each eServer contains 8 DCMs (16 cores), i.e. total node L3 cache is 144 MB.
http://www.hpcx.ac.uk/support/documentation/UserGuide/HPCxuser/Architecture_Overview.html
[15] For IBM eServer we used LAPACK and threaded BLAS from ESSL.

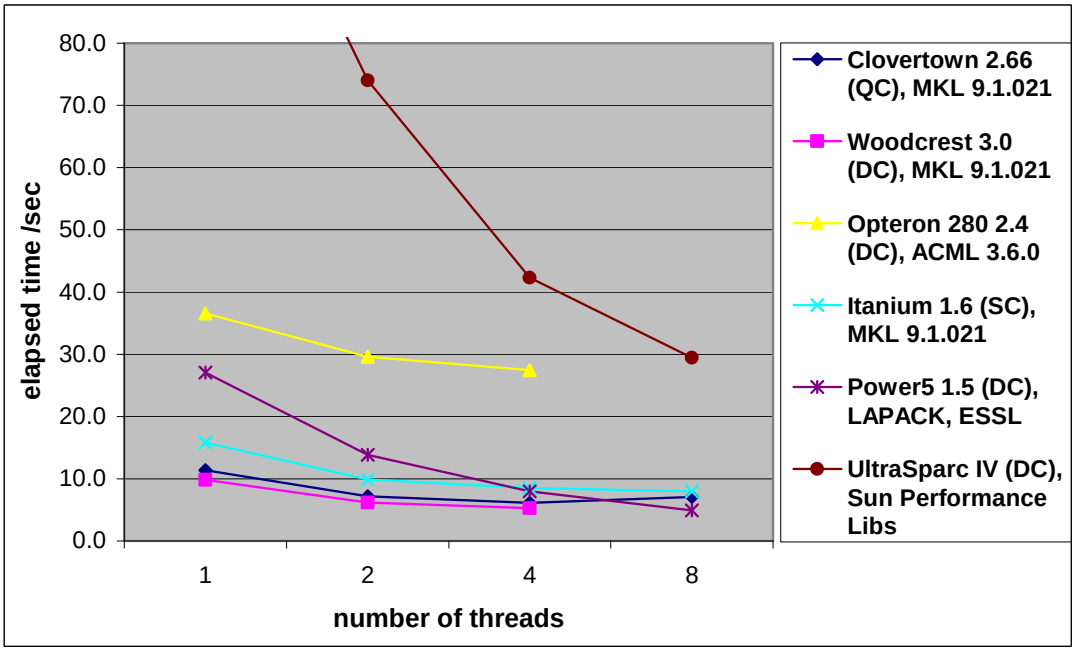

Figure 5: Threaded performance of DSYEV: matrix size is 3537, all eigenvalues are computed, no eigenvectors.

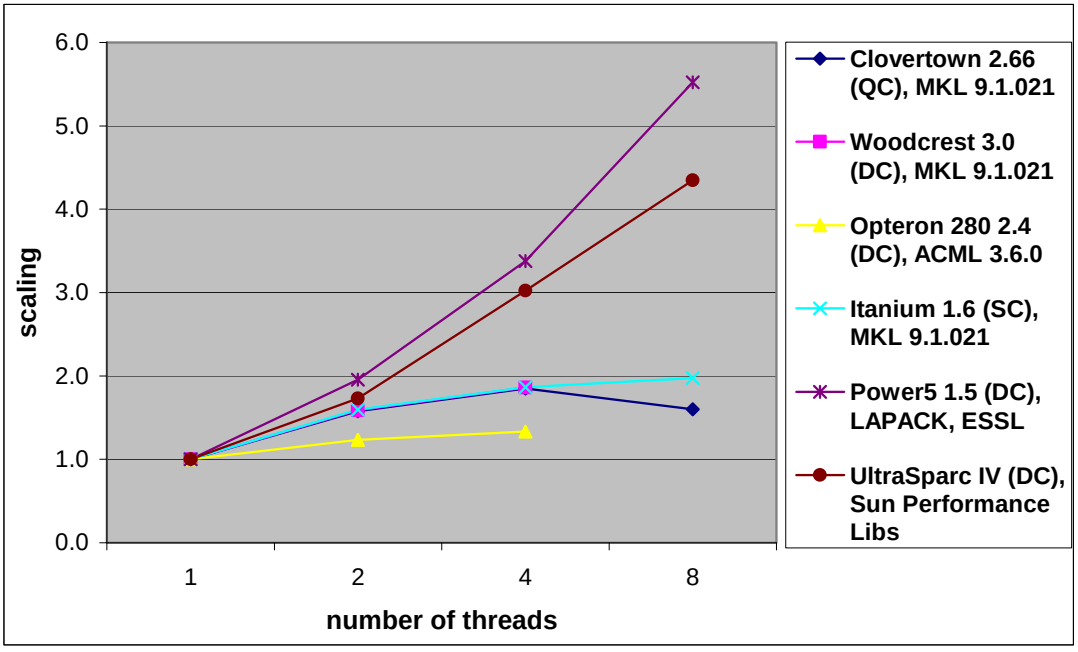

Figure 6: Scaling of DSYEV: matrix size is 3537, all eigenvalues are computed, no eigenvectors (i.e. same as in Figure 5).

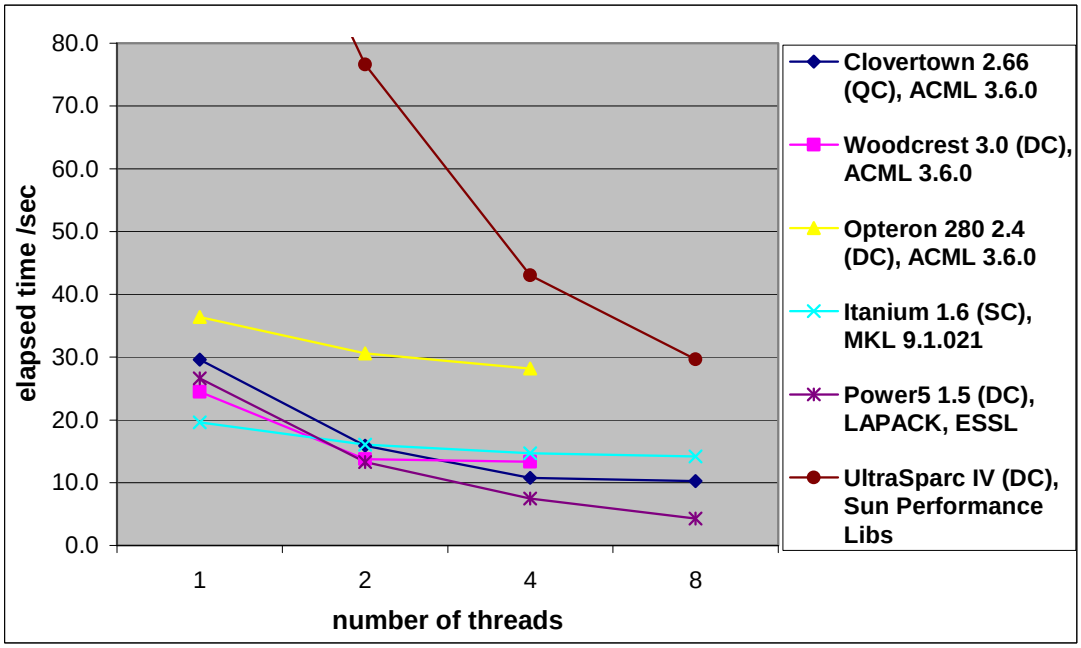

Figure 7: Threaded performance of DSYEVR: matrix size is 3537, 1% of eigenvectors is computed.

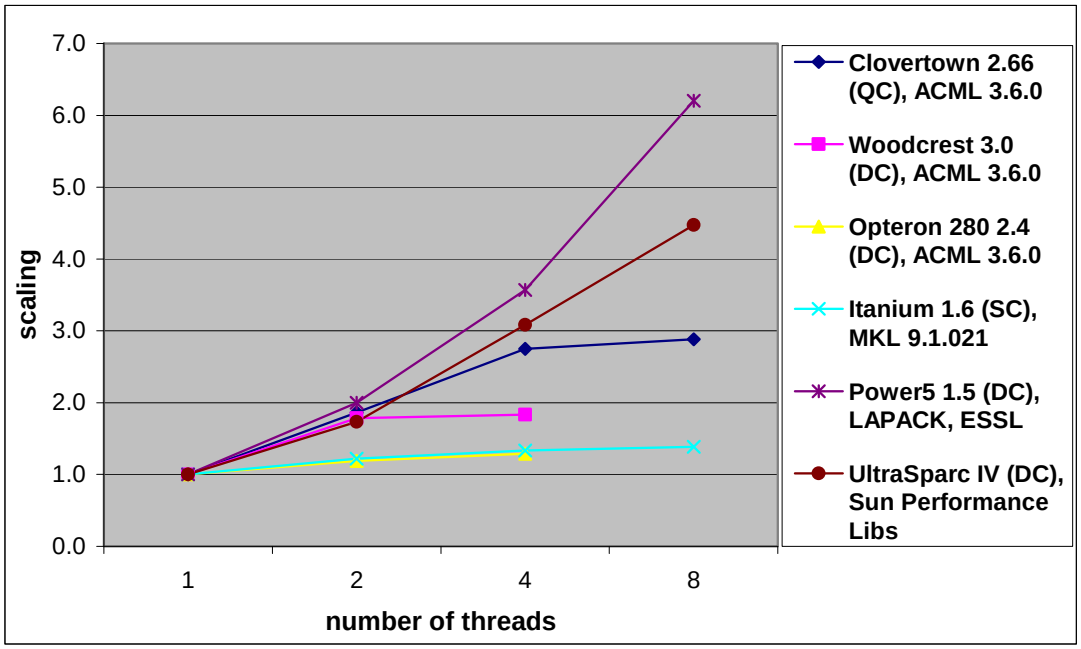

Figure 8: Scaling of DSYEVR: matrix size is 3537, 1% of eigenvectors is computed (i.e. same as in Figure 6Figure 7).

Conclusions

We have studied the performance of symmetric diagonalisers for dense matrices which are important in many applications and are part of the Linear Algebra PACKage (LAPACK). We have investigated the performance of different libraries on the latest breed of Intel's x86 processors and saw very good serial performance, and the recent introduction of fused multiply-add means these platforms often beat Itanium. Due to the late delivery of the AMD Barcelona quad-core platform, we have not yet been able to carry out an investigation of this technology but hope to report on its performance in the near future.

When it comes to practical choices, if eigenvectors are not required then DSYEV is recommended due to its superior performance. However if eigenvectors are needed then DSYEVX or DSYEVR should be used. An additional factor for consideration might be the required working space for these routines – something we have not thus far considered. Another interesting issue this study has highlighted is the surprisingly good threaded performance of ACML on Intel platforms. Although ACML is slower than MKL on a single thread it can be certainly recommended for multiple threads.

With the advent of multi-core processors library writers are under pressure to produce efficient parallel implementations more than ever. We have looked into the efficiency of threaded LAPACK libraries and identified the bottleneck as typically being in the tri-diagonalisation step. This step is memory bandwidth bounded and therefore severely limits the overall performance even if using a fully threaded LAPACK.

Further algorithmic advances are needed if we are to use all the cores efficiently, and the number of cores in a socket is likely to double every 18-24 months in keeping with Moore's Law. Although we did notice some substantial improvements in performance of LAPACK libraries (for example, DSYEV was much improved in MKL 9.1.021 versus 9.1.018) and the diversity of current choice ranging from commercial libraries to ATLAS and GotoBLAS can only be welcomed, our analysis showed that multi-core processors are not very efficient when using current LAPACK algorithms and that this situation will only get worse in the coming years with current programming models.

In fact there are (probably) insurmountable scalability issues for standard Pthread/OpenMP programming models due to the fact that using locks to implement synchronization is inherently a bottleneck. We envisage alternative approaches to concurrency such as software and/or hardware implementations of the transactional memory model becoming available. Indeed, Sun have recently announced that its multi-threaded Rocks platform (16 cores and 32 threads) will offer some support for this approach in hardware early in 2008. How easy it will be to transition legacy codes and libraries to this and any other alternative approaches remains to be seen.

Acknowledgements

We would like to thank Intel for making Woodcrest and Clovertown workstations available to us and Victor Gamayunov (Intel) for hpcm tool. Also we are grateful to Andrey Naraikin and Andrey Shemyakin (Intel) for providing early access to MKL, Bruno Silva (UCL) for running dbench on SunFire SMP machine, Andy Sunderland (STFC) for running dbench on HPCx, and Kazushige Goto (TACC) for advice on his library.

Appendix

Table 3: Performance of different LAPACK libraries on Intel Clovertown 2.66 GHz, matrix size 7075.

vectors	sub	n	MKL 9.1.021	MKL 9.1.018	LAPACK 3.1.1	LAPACK 3.1.1 + GotoBLAS 1.16	ACML 3.6.0
100% spectrum							
n	DSYEV	1	88.0	217.6	281.4	232.9	254.5
n	DSYEV	2	50.4	186.2		217.8	149.5
n	DSYEV	4	40.7	176.1		211.4	131.3
n	DSYEV	8	47.5	175.0		211.0	131.4
n	DSYEVD	1	214.9	217.6	282.0	232.9	254.5
n	DSYEVD	2	185.2	185.9		217.9	149.5
n	DSYEVD	4	176.2	176.2		211.4	131.4
n	DSYEVD	8	174.6	174.9		211.0	131.5
n	DSYEVX	1	214.7	217.6	280.4	232.9	254.5
n	DSYEVX	2	185.1	185.7		217.8	149.5
n	DSYEVX	4	175.7	176.1		211.4	131.4
n	DSYEVX	8	174.6	174.8		211.0	131.5
n	DSYEVR	1	215.9	219.6	279.3	233.9	278.6
n	DSYEVR	2	185.8	186.5		218.3	161.0
n	DSYEVR	4	176.6	177.0		212.0	136.1
n	DSYEVR	8	175.4	175.7		211.5	132.9
v	DSYEV	1	516.1	524.9	2138.9	1659.7	1244.0
v	DSYEV	2	349.5	352.9		1611.1	658.7
v	DSYEV	4	282.3	286.3		1592.4	398.7
v	DSYEV	8	270.6	274.2		1591.4	272.9
v	DSYEVD	1	336.0	344.4	1115.1	357.1	417.9
v	DSYEVD	2	248.1	252.7		285.8	242.9
v	DSYEVD	4	216.9	219.2		257.6	error
v	DSYEVD	8	211.0	214.6		252.6	error
v	DSYEVX	1	214.8	217.4	282.5	233.0	254.5
v	DSYEVX	2	184.8	185.8		217.9	149.5
v	DSYEVX	4	175.7	176.0		211.4	131.4
v	DSYEVX	8	174.7	174.8		211.0	131.4
v	DSYEVR	1	215.8	219.8	279.7	234.0	278.5
v	DSYEVR	2	185.5	186.5		218.2	160.9
v	DSYEVR	4	176.6	176.9		212.0	136.1
v	DSYEVR	8	174.9	175.6		211.5	132.8
1% spectrum							
n	DSYEVX	1	213.1	215.8	281.1	231.3	253.2
n	DSYEVX	2	183.1	184.0		216.4	147.8
n	DSYEVX	4	174.0	174.3		209.7	129.5
n	DSYEVX	8	173.0	173.2		209.2	129.6
n	DSYEVR	1	214.2	218.0	280.9	232.3	253.0
n	DSYEVR	2	183.8	184.7		216.5	147.8
n	DSYEVR	4	174.9	175.2		210.1	129.4
n	DSYEVR	8	173.6	173.6		209.9	129.3
v	DSYEVX	1	213.3	215.9	281.7	231.3	252.7
v	DSYEVX	2	183.1	184.1		216.2	147.6
v	DSYEVX	4	174.0	174.3		209.8	129.4
v	DSYEVX	8	172.8	173.1		209.3	129.5
v	DSYEVR	1	214.2	218.1	280.0	232.5	252.7
v	DSYEVR	2	183.8	184.7		216.5	147.7
v	DSYEVR	4	175.3	175.2		210.3	129.5
v	DSYEVR	8	173.6	174.1		209.9	129.5

Table 4: Performance of LAPACK libraries on Intel Clovertown 2.66 GHz, matrix size 3537.

vectors	sub	n	MKL 9.1.021	MKL 9.1.018	LAPACK 3.1.1	LAPACK 3.1.1 + GotoBLAS 1.16	ACML 3.6.0
100% spectrum							
n	DSYEV	1	11.4	26.9	34.7	29.5	30.0
n	DSYEV	2	7.2	23.4		27.8	16.3
n	DSYEV	4	6.1	22.0		27.0	11.2
n	DSYEV	8	7.1	21.7		26.8	10.8
n	DSYEVD	1	26.5	26.9	34.7	29.5	30.0
n	DSYEVD	2	23.3	23.3		27.8	16.3
n	DSYEVD	4	22.0	22.0		27.0	11.2
n	DSYEVD	8	21.7	21.7		26.8	10.8
n	DSYEVX	1	26.6	26.8	34.6	29.5	30.0
n	DSYEVX	2	23.3	23.3		27.8	16.3
n	DSYEVX	4	21.9	22.0		26.9	11.3
n	DSYEVX	8	21.8	21.8		26.8	10.8
n	DSYEVR	1	26.7	27.0	34.6	29.6	36.3
n	DSYEVR	2	23.4	23.4		27.8	19.3
n	DSYEVR	4	22.0	22.1		27.0	12.5
n	DSYEVR	8	21.8	21.8		26.7	11.1
v	DSYEV	1	65.9	66.7	272.5	213.4	134.7
v	DSYEV	2	45.9	46.0		207.9	69.1
v	DSYEV	4	37.2	37.6		205.6	38.3
v	DSYEV	8	35.1	36.2		205.3	25.4
v	DSYEVD	1	44.8	45.8	153.9	47.6	51.0
v	DSYEVD	2	33.1	33.5		38.4	29.0
v	DSYEVD	4	28.5	28.8		34.7	20.5
v	DSYEVD	8	27.5	27.8		33.8	19.0
v	DSYEVX	1	26.6	26.9	34.6	29.5	30.0
v	DSYEVX	2	23.3	23.3		27.8	16.3
v	DSYEVX	4	21.9	22.0		26.9	11.2
v	DSYEVX	8	21.7	21.7		26.7	10.8
v	DSYEVR	1	26.6	27.0	34.7	29.6	36.3
v	DSYEVR	2	23.4	23.5		27.8	19.3
v	DSYEVR	4	22.0	22.1		26.9	12.5
v	DSYEVR	8	21.8	21.8		26.7	11.1
1% spectrum							
n	DSYEVX	1	26.1	26.4	34.6	29.1	29.6
n	DSYEVX	2	22.8	22.9		27.3	15.9
n	DSYEVX	4	21.5	21.5		26.5	10.8
n	DSYEVX	8	21.2	21.3		26.3	10.3
n	DSYEVR	1	26.3	26.6	34.6	29.2	29.6
n	DSYEVR	2	22.9	22.9		27.6	15.9
n	DSYEVR	4	21.6	21.6		26.5	10.8
n	DSYEVR	8	21.4	21.3		26.3	10.3
v	DSYEVX	1	26.0	26.4	34.6	29.1	29.6
v	DSYEVX	2	22.8	22.9		27.4	15.9
v	DSYEVX	4	21.4	21.5		26.5	10.8
v	DSYEVX	8	21.3	21.2		26.3	10.3
v	DSYEVR	1	26.3	26.6	34.6	29.2	29.6
v	DSYEVR	2	22.9	23.0		27.4	15.9
v	DSYEVR	4	21.6	21.6		26.5	10.8
v	DSYEVR	8	21.4	21.4		26.3	10.3

Table 5: Performance of LAPACK libraries on different hardware, matrix size 7075.

vectors	sub	n	Clovertown 2.66 (QC), MKL 9.1.021, ACML 3.6.0 (Intel)	Woodcrest 3.0 (DC), MKL 9.1.021, ACML 3.6.0 (Intel)	Opteron 280 2.4 (DC), ACML 3.6.0 (Intel)	Itanium 1.6 (SC), MKL 9.1.021	Power5 1.5(DC), LAPACK + ESSL BLAS
100% spectrum					**timing**		
n	DSYEV	1	88.0	75.4	290.1	124.9	247.6
n	DSYEV	2	50.4	42.9	249.5	90.3	133.1
n	DSYEV	4	40.7	35.0	250.5	72.7	64.6
n	DSYEV	8	47.5			57.0	31.8
n	DSYEVD	1	254.5	171.0	287.0	150.4	247.7
n	DSYEVD	2	149.5	145.0	249.7	124.8	132.2
n	DSYEVD	4	131.4	138.3	250.2	109.5	64.6
n	DSYEVD	8	131.5			103.6	31.6
n	DSYEVX	1	254.5	170.9	287.0	150.4	249.4
n	DSYEVX	2	149.5	145.0	249.4	123.0	132.4
n	DSYEVX	4	131.4	138.6	250.9	154.2	64.9
n	DSYEVX	8	131.5			103.3	31.7
n	DSYEVR	1	278.6	170.9	305.3	153.6	250.0
n	DSYEVR	2	161.0	145.4	252.3	127.2	134.4
n	DSYEVR	4	136.1	139.4	254.7	111.9	65.4
n	DSYEVR	8	132.9			104.5	31.8
v	DSYEV	1	516.1	432.5	1350.2	594.2	1893.9
v	DSYEV	2	349.5	287.3	734.1	484.8	1739.3
v	DSYEV	4	282.3	237.1	531.1	304.7	
v	DSYEV	8	270.6			214.0	
v	DSYEVD	1	336.0	275.9	495.5	316.5	463.7
v	DSYEVD	2	248.1	199.3	372.9	220.8	243.2
v	DSYEVD	4	216.9	178.4	343.7	194.8	124.5
v	DSYEVD	8	211.0			202.6	65.3
v	DSYEVX	1	254.5	171.0	287.2	150.4	248.5
v	DSYEVX	2	149.5	144.8	249.6	121.4	133.2
v	DSYEVX	4	131.4	138.7	251.0	109.5	64.8
v	DSYEVX	8	131.4			104.0	31.7
v	DSYEVR	1	278.5	170.6	304.9	153.8	250.0
v	DSYEVR	2	160.9	145.2	258.9	123.1	134.2
v	DSYEVR	4	136.1	139.4	256.1	111.0	65.3
v	DSYEVR	8	132.8			105.0	31.9
1% spectrum							
n	DSYEVX	1	253.2	169.5	285.7	149.3	245.7
n	DSYEVX	2	147.8	143.4	249.8	120.5	131.5
n	DSYEVX	4	129.5	137.0	250.3	107.6	62.4
n	DSYEVX	8	129.6			101.4	29.4
n	DSYEVR	1	253.0	169.2	287.5	152.4	247.8
n	DSYEVR	2	147.8	143.9	250.6	124.8	132.1
n	DSYEVR	4	129.4	137.9	252.6	108.2	63.2
n	DSYEVR	8	129.3			102.9	29.5
v	DSYEVX	1	252.7	169.3	286.0	149.3	245.7
v	DSYEVX	2	147.6	143.4	248.6	131.4	131.1
v	DSYEVX	4	129.4	137.1	253.6	107.1	62.4
v	DSYEVX	8	129.5			102.1	29.3
v	DSYEVR	1	252.7	169.2	286.4	152.4	247.9
v	DSYEVR	2	147.7	143.7	248.9	144.8	132.3
v	DSYEVR	4	129.5	138.0	250.6	108.4	62.9
v	DSYEVR	8	129.5			103.1	29.6

Table 6: Performance of LAPACK libraries on different hardware, matrix size 3537.

vectors	sub	n	Clovertown 2.66 (QC), MKL 9.1.021, ACML 3.6.0 (intel)	Woodcrest 3.0 (DC), MKL 9.1.021, ACML 3.6.0 (intel)	Opteron 280 2.4 (DC), ACML 3.6.0 (Intel)	Itanium 1.6 (SC), MKL 9.1.021	Power5 (DC), LAPACK + ESSL BLAS	UltraSparc IV (DC), Sun Performance Libs
100% spectrum					**timing**			
n	DSYEV	1	11.4	9.8	36.6	15.8	27.1	128.0
n	DSYEV	2	7.2	6.2	29.6	9.9	13.8	74.0
n	DSYEV	4	6.1	5.3	27.4	8.5	8.0	42.3
n	DSYEV	8	7.1			8.0	4.9	29.4
n	DSYEVD	1	30.0	24.9	36.3	19.5	27.1	127.7
n	DSYEVD	2	16.3	14.1	29.7	16.3	13.8	74.3
n	DSYEVD	4	11.2	13.8	27.4	14.9	8.0	42.1
n	DSYEVD	8	10.8			14.4	4.9	29.8
n	DSYEVX	1	30.0	24.8	36.5	19.5	27.1	128.0
n	DSYEVX	2	16.3	14.1	29.6	16.3	13.8	74.2
n	DSYEVX	4	11.3	13.8	27.2	14.9	8.0	42.3
n	DSYEVX	8	10.8			14.3	4.9	29.7
n	DSYEVR	1	36.3	30.4	41.1	19.9	27.2	155.7
n	DSYEVR	2	19.3	16.8	32.1	16.4	13.9	99.4
n	DSYEVR	4	12.5	14.9	28.5	15.1	8.0	66.1
n	DSYEVR	8	11.1			14.7	4.9	53.0
v	DSYEV	1	134.7	55.2	167.9	91.4	237.2	1556.1
v	DSYEV	2	69.1	37.6	98.0	56.5	214.4	795.2
v	DSYEV	4	38.3	31.0	62.6	38.9	204.4	496.0
v	DSYEV	8	25.4			28.9	200.1	334.8
v	DSYEVD	1	51.0	43.07	63.5	43.9	57.6	227.9
v	DSYEVD	2	29.0	24.99	47.1	29.9	29.6	126.6
v	DSYEVD	4	20.5	21.01	40.5	23.3	16.8	70.7
v	DSYEVD	8	19.0			20.8	10.6	46.5
v	DSYEVX	1	30.0	24.85	36.6	19.5	27.2	128.0
v	DSYEVX	2	16.3	14.12	30.7	16.3	13.8	74.6
v	DSYEVX	4	11.2	13.78	28.8	15.0	8.0	42.3
v	DSYEVX	8	10.8			14.4	4.9	29.7
v	DSYEVR	1	36.3	30.44	41.4	20.0	27.2	155.8
v	DSYEVR	2	19.3	16.82	33.2	16.5	13.9	99.5
v	DSYEVR	4	12.5	14.86	29.4	15.1	8.0	66.2
v	DSYEVR	8	11.1			15.1	4.9	53.0
1% spectrum								
n	DSYEVX	1	29.6	24.47	36.4	19.2	26.5	127.0
n	DSYEVX	2	15.9	13.71	30.5	15.8	13.3	73.5
n	DSYEVX	4	10.8	13.34	28.3	14.5	7.4	41.3
n	DSYEVX	8	10.3			14.2	4.3	28.8
n	DSYEVR	1	29.6	24.47	36.4	19.7	26.7	132.7
n	DSYEVR	2	15.9	13.74	30.7	16.0	13.3	76.6
n	DSYEVR	4	10.8	13.34	28.3	22.0	7.5	43.2
n	DSYEVR	8	10.3			14.2	4.3	29.9
v	DSYEVX	1	29.6	24.43	36.7	19.2	26.5	127.1
v	DSYEVX	2	15.9	13.71	30.7	15.9	13.2	73.4
v	DSYEVX	4	10.8	13.34	28.3	14.5	7.4	41.4
v	DSYEVX	8	10.3			14.3	4.3	28.6
v	DSYEVR	1	29.6	24.47	36.4	19.6	26.6	132.6
v	DSYEVR	2	15.9	13.73	30.6	16.0	13.3	76.6
v	DSYEVR	4	10.8	13.35	28.2	14.7	7.5	43.0
v	DSYEVR	8	10.3			14.2	4.3	29.6