

Applying SUD Parallelism to improve parallel efficiency of ESI-Group PAM-Crash v2007 on multi-core processors

Michael Riedmann michael.riedmann@hp.com
HP European Performance Center, Böblingen, Germany
Revision 1.1, March 2008

Abstract	2
Motivation.....	2
Situation in chip manufacturing	2
Challenges to software development.....	2
Amdahl's law revisited	3
Scope and relevance	3
The world of multi core processors	4
Description of Intel based nodes (HP BladeSystem BL460c).....	4
Description of AMD based nodes (HP BladeSystem BL465c).....	5
Summarizing the evolution from DC to QC processor generations	6
Discussion of some optimization approaches	6
Goal definition.....	6
Overlay of computation and file I/O a.k.a. Async I/O	6
Overlay of computation and communication	6
Undersubscription.....	7
Combining distributed and shared memory parallelism	7
Definition of measurements.....	8
Primary measurements on Intel based nodes	8
Supplementary Measurements on Intel based nodes.....	11
PAM-Crash DMP versus SUD performance	12
Performance characterization.....	12
Building a SUD version	12
Selection of test cases	12
Neon Measurements.....	14
Q2 Measurements.....	15
SMD Measurements.....	17
PST Measurements.....	19
BMI Measurements	21
Q5 Measurements.....	23
Testing SUD on Dual-Core nodes.....	25
Summary of PAM-Crash performance on Intel processors	26
Appendix.....	27
References.....	27
Credits.....	27
For more information	27

Abstract

This investigation is part of the multi core optimization program initiated by HP's High Performance Computing Division. The purpose of this program is to explore ways to best utilize the power of multi core processors with HPC applications.

The two HPC applications, one industrial and one scientific, which were chosen for this investigation, have quite different performance characteristics. ESI-Group's PAM-Crash is known for its moderate memory load. It scales well with clock speed and cache size. The COSMO LM_RAPS weather model is more demanding on memory bandwidth but is still sensitive to cache size.

The goal was to tune both applications to achieve the best possible speedup when migrating from a dual core processor platform to a quad core processor with equal clock rates. This is the most common challenge today.

This paper covers PAM-Crash only. The LM_RAPS part is covered by a separate paper [2].

Most current and all future multi core processors are sharing caches between multiple cores. Two major approaches were tried to exploit this. First, the effect of undersubscription was measured, the case where only half of the cores but all of the cache and bus resources are utilized. The second approach was to combine DMP and SMP parallelism, called SUD, specifically the case when SMP threads are placed on common caches.

The result is quite positive for both applications. The cumulated performance gain for PAM-Crash is in the range of 1.3 to 1.7. The cumulated performance gain for LM_RAPS is around 1.3.

A number of effects contribute to these performance gains: First, the additional HW resources like the doubled number of cores and cache size. Then, application usage is optimized by explicit process placement. Finally there is a benefit from SUD parallelism.

All measurements were conducted on an Intel Xeon based system. They will be repeated on AMD based quad core systems whenever they become available.

Motivation

Situation in chip manufacturing

Progress of processor manufacturing has apparently hit physical limits that inhibit further growth of clock frequencies. Chipmakers have changed direction accordingly. Instead of clock rate increases they now increase the number of processing units on a single chip. These processing units are called cores; the chips are called multi core processors.

This approach sustains the illusion of Moore's law delivering never-ending growth of compute capacity. In the November 2007 Top500 list the positions 3, 4 and 5 are held by clusters based on such multi core processors.

Challenges to software development

Increases of processor clock rate usually speeds up any HPC application without extra effort. But an increased number of cores per chip at the same clock rate does not speed up sequential applications at all. Nor does it help applications that have already reached the limits of their parallel efficiency. Only those applications that have excellent parallel efficiency can further exploit the power of multi core processors. So the challenges to software developers are:

- To start parallelizing sequential applications.
This challenge applies mainly to desktop applications where parallelism is not widely used.
- To work on improving efficiency and scalability of parallel applications.
Developers have to revisit their approaches to parallelism and pay more attention to the remaining sequential portions in the code. This will also require better knowledge of the

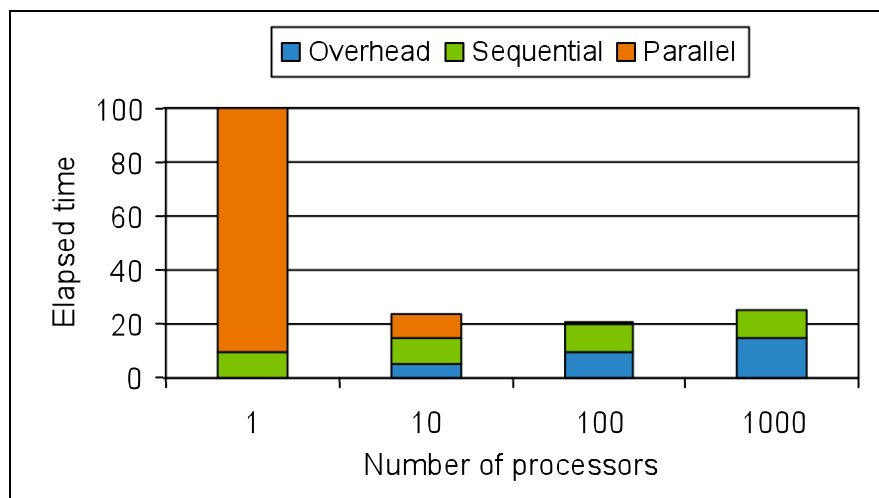
system architecture. Optimizing for multi core processors is like optimizing for vector processors: Only by adjustments in the source code can the full power be exploited.

Amdahl's law revisited

Whenever a parallel program is tested with twice the usual number of processors then some common observations are:

- The sequential portions of the code become more significant. That is particularly true for file I/O.
- The overhead of parallelization increases. This includes all extra tasks like domain decomposition, message passing, load balancing, and access of shared resources. In many common cases the parallel overhead increases logarithmically with the number of processors.

Figure 1. Example for scalability of a 90% efficient parallel application



Scope and relevance

Choice of applications

PAM-Crash was chosen simply because the author knows it very well and has conditional access to the source code and its developers. Even more relevant is that this application is widely used in its target market. PAM-Crash is used in the automotive and aircraft industry for impact, crash and passenger safety simulation.

The world of multi core processors

Description of Intel based nodes (HP BladeSystem BL460c)

Figure 2. HP BladeSystem BL460c Node with DC Processors

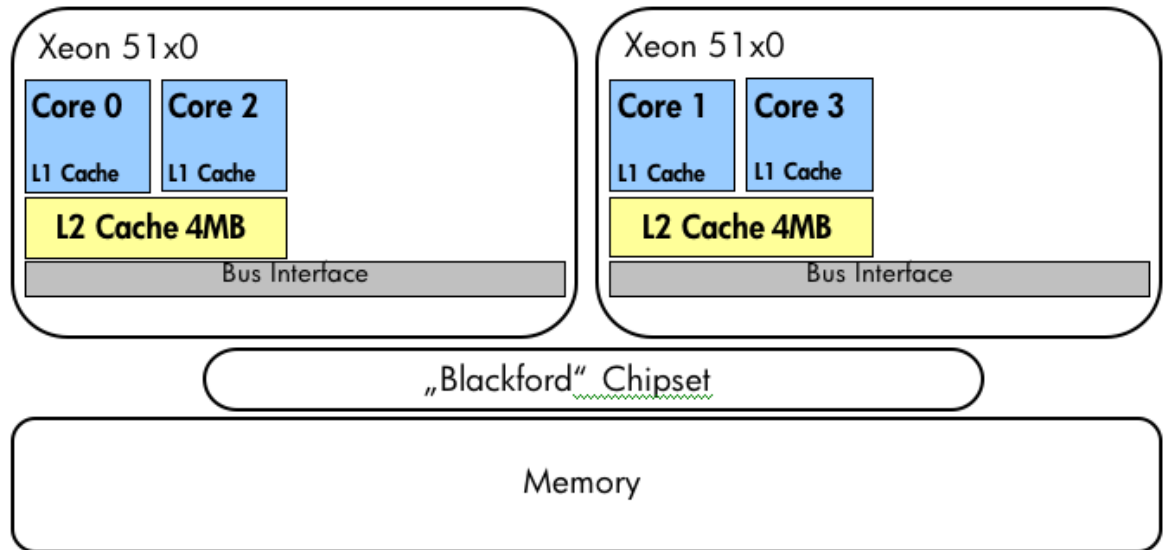
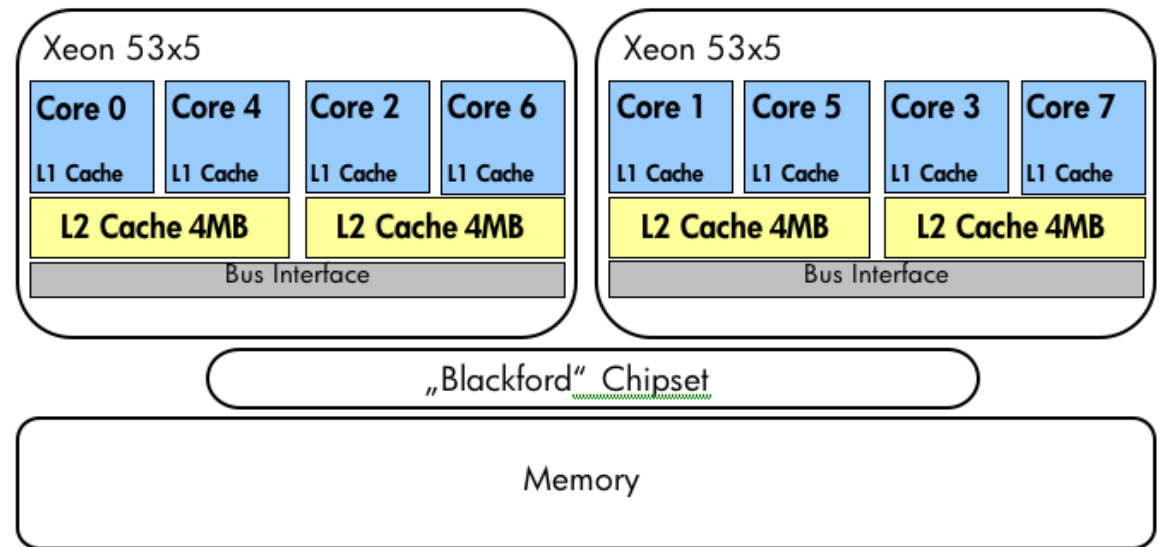


Figure 3. HP BladeSystem BL460c Node with QC Processors



Notes on Intel based DC and QC nodes

- UMA architecture with all memory access going through the chip set.
- L2 caches are shared between 2 cores.
- L2 cache size is 2MB per core in both systems. The doubled number of cores also doubles the total cache size.

- Cumulated bus bandwidth is identical in both systems. This means that QC systems have only half the bus bandwidth per core so that performance issues may arise for memory intensive applications.

Description of AMD based nodes (HP BladeSystem BL465c)

Figure 4. HP BladeSystem BL465c Node with DC Processors

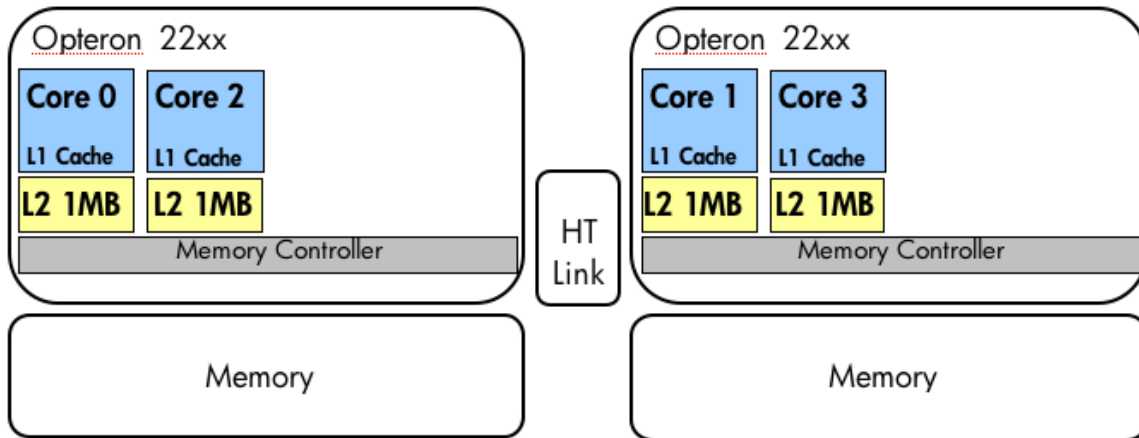
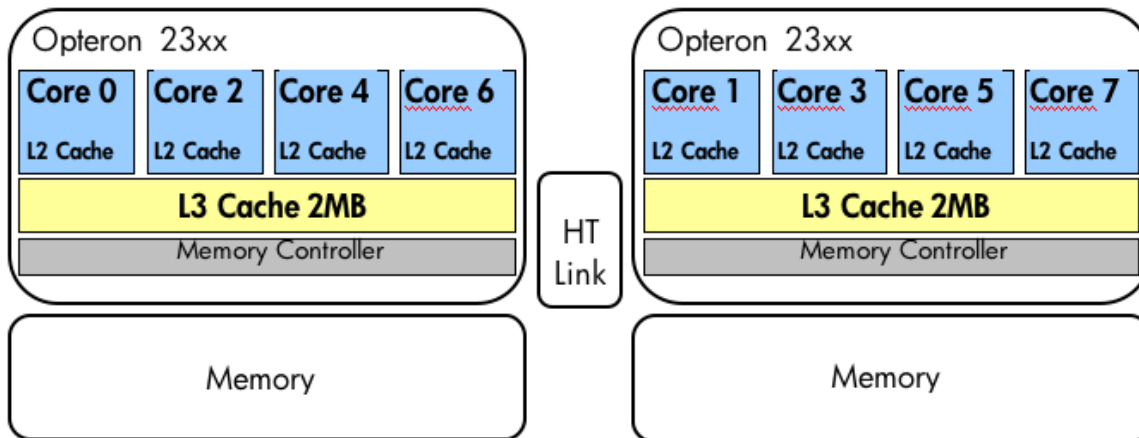


Figure 5. HP BladeSystem BL465c Node with QC Processors (future product)



Notes on AMD based DC and QC nodes

- NUMA architecture with physically separated memory pools
- No cache sharing in DC processors, L3 caches are shared between all cores in QC processors.
- Cumulated cache size is 1MB per core in both systems. The doubled number of cores also doubles the total cache size.
- Cumulated bus bandwidth is identical in both systems. This means that QC systems have only half the bus bandwidth per core so that performance issues may arise for memory intensive applications.

Summarizing the evolution from DC to QC processor generations

The comparison of previous generation DC processors with the current QC generation can be summarized as follows.

The bad news

- Clock rates have not increased.
- Bus bandwidth per core is divided by 2.

The good news

- The number of cores per chip and per system has doubled.
- The cache size per core is equal or has increased.
- L2 or L3 caches are shared between cores.

The challenges

- How to utilize the additional cores?
- How to work around the bus bandwidth reduction?
- How to handle shared caches?

Discussion of some optimization approaches

Goal definition

The goal is tuning the two applications to achieve the best possible speedup when migrating from a dual core processor platform to a quad core processor with equal clock rates. This is the most common challenge today.

Overlay of computation and file I/O a.k.a. Async I/O

Having abundant cores allows to even assign “part time” tasks to dedicated cores. The most common example is asynchronous file I/O, e.g. output of restart files. These tasks might be idle 95% of their time but they can reduce the percentage of sequential execution as referenced by Amdahl’s law.

This can be implemented explicitly within the application or it can be done by the operating system. The most common approach in HPC is to implement it in the application as this avoids OS dependencies and is likely to be more efficient.

Overlay of computation and communication

This idea can be applied to a small set of message passing applications. Assigning dedicated resources to propagate messages makes sense whenever immediate synchronization is not required. This means that a process has independent work to do while messages are propagated.

It can be implemented explicitly by using OpenMP or POSIX threads. In that case one or more threads would continue to compute while another dedicated thread would do nothing but propagating messages.

Implicit implementation can be realized within the messages passing interface. A great example is the capability of HP-MPI, which can assign its activity to a dedicated thread. This behavior can be controlled from the environment and requires no source code modifications. Effective implicit overlay requires that messages are passed by nonblocking API calls, e.g. by using `MPI_Isend` and `MPI_Irecv` and by placing the corresponding `MPI_Waitall` call as late as possible. Unfortunately very few applications are written that way so that even implicit overlay requires source modifications before it can become efficient.

Undersubscription

Undersubscription means that not all available cores are used. This approach is appropriate when bus bandwidth is clearly insufficient to keep all cores busy. The positive effect of undersubscription in combination with shared L2 caches is that idle cores yield their share of the L2 cache to the busy cores.

This way it is possible to have a QC system perform significantly better than a DC system by just doubling the effective cache size per busy core. It requires that compute processes are placed and bound to cores appropriately.

Elegance is something different. One has to be quite desperate to work this way. Besides the extra cache no other components of the processor are utilized. So this is a “better than nothing” type of approach.

Combining distributed and shared memory parallelism

Almost forgotten: SMP

Shared Memory Programming (SMP) has gone out of fashion in recent years for a number of reasons.

1. SMP was often used for quick and dirty parallelization of legacy codes. That approach yields only moderate scalability as long as no major effort is spent for true reengineering. Consequently SMP got the reputation of being somewhat inefficient.
2. Some supercomputer architectures are purely DMP capable and unable to run SMP applications.
3. The promise of cheap clusters made of thin industry standard servers has led many developers to exclusively go for Distributed Memory Programming (DMP) and abandon the initial SMP versions.
4. The most popular types of cluster nodes used to have no more than 1 or 2 processors so there was no point in considering SMP.
5. All future implementations of major processor types like Intel Xeon, Intel Itanium and AMD Opteron will be NUMA architectures. SMP parallelism as done with OpenMP is difficult to optimize for memory locality on NUMA systems.

The downsides of DMP

DMP yields good scalability for many applications. But there are tradeoffs and costs.

- Efficient DMP requires complete reengineering of the source code.
- DMP introduces extra overhead for domain decomposition and message passing. The overhead usually grows with the number of domains.
- The quality of domain decompositions gets worse when the number of domains is increased and thus introduces load imbalance. Dynamic load balancing e.g. domain resizing at runtime is extremely difficult to implement. In many applications this is not available at all.
- DMP process data spaces are disjoint. There is no way to utilize shared caches.

How can SMP underneath DMP help ?

SMP can be used to parallelize within a node, within a multi core processor, within the scope of a shared cache or within the scope of a NUMA node. DMP can be used on top of that to parallelize across node boundaries.

- SMP reduces the number of DMP domains. That can improve load balance and reduce DMP overhead.
- With SMP it is easier to parallelize those leftover portions of the code that are still sequential.
- SMP allows thread placement on cores with shared caches. As threads have common data spaces this can improve cache efficiency.

Let's call it SUD

The acronym SUD is used to describe a hybrid SMP under DMP version of the chosen applications.

Definition of measurements

Description of Intel based nodes

Cluster nodes

All subsequent tests were conducted on clusters of HP BladeSystem BL460c nodes. These nodes are equipped with 3.0 GHz processors, with either DC (Xeon 5160) or QC (Xeon 5365) chips. This platform was chosen because besides the processor all other system components are identical. Even the cache size per core is identical. The chipset is Intel "Blackford" with 2 busses connecting to each processor. The bus is clocked at 1333M transactions/sec.

Cluster interconnect

The cluster interconnect is made of latest generation Mellanox "ConnectX" InfiniBand HCA's. These are connected to a 16-port InfiniBand switch blade within the blade enclosure. That switch blade has 8 uplinks to the backbone switch. The entire InfiniBand infrastructure is running at 4X Double Data Rate, which makes 20 Mbits/sec.

Detection of core numbering

The numbering scheme for the cores as seen by the OS depends on BIOS settings and can differ between HW vendors even if the same processor type is used. In addition there are different numbering schemes for different OS's like Linux and Windows.

In the absence of reliable documentation it is recommended to use tools like Intel's cpuinto, which is part of Intel MPI but can be used standalone. The output of cpuinto for Intel based BL460c cluster nodes looks as follows:

```
Architecture : x86_64
Hyperthreading: disabled
Packages     : 2
Cores       : 8
Processors  : 8
==== Processor identification ====
Processor    Thread  Core  Package
0            0      0     0
1            0      0     1
2            0      2     0
3            0      2     1
4            0      1     0
5            0      1     1
6            0      3     0
7            0      3     1
==== Processor placement ====
Package Cores          Processors
0       0,2,1,3        0,2,4,6
1       0,2,1,3        1,3,5,7
==== Cache sharing ====
Cache  Size          Processors
L1     32 KB         no sharing
L2     4 MB         (0,4) (1,5) (2,6) (3,7)
```

Explicit process placement with HP-MPI

HP-MPI provides a comprehensive set of process placement options through the

```
-cpu_bind=<options>
```

command line switch. Details are outlined in the man-page for mpirun and in the HP-MPI release notes.

Primary measurements on Intel based nodes

The following measurements are designed for all DC and QC systems having 2 cores connected to a shared cache. This fits all current and near future Intel Xeon processors.

The Base: DMP on Dual Core

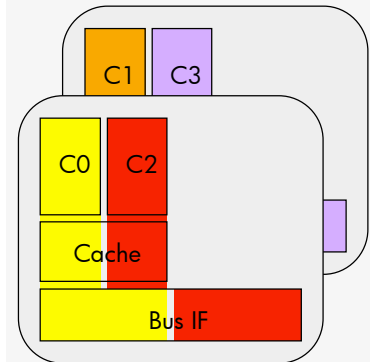
The overall goal is to tune for best possible performance gain of QC processors over DC processors. So the standard DMP version on DC processors defines the base for all subsequent tests.

HP-MPI Settings:

```
-np 4 -cpu_bind=v,rank
```

Process Placement

Resource Usage



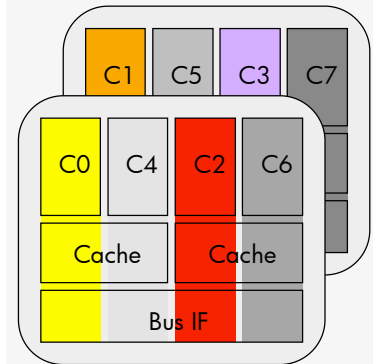
Step 1: DMP on Quad Core

This measurement uses the same number of nodes with QC instead of DC processors. So it quantifies the performance gain that is achieved by the doubled number of cores and cache size. It can be considered the pure hardware related gain factor. Depending on application and test case this can be below 1.

HP-MPI Settings:

```
-np 8 -cpu_bind=v,rank
```

Process Placement Resource Usage



Step 2: DMP with Undersubscription on Quad Core

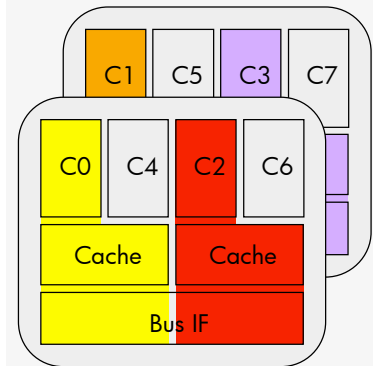
Undersubscription is the next approach, which still does not require source code modifications. It can be easily applied to any application, as it only requires correct process placement. This test uses only half of the cores but all of the cache. It can as well be considered a dual core test with doubled cache size.

The ratio between the previous (Step 1) and this timing quantifies the gain factor that can be attributed to undersubscription.

HP-MPI Settings:

```
-np 4 -cpu_bind=v,rank
```

Process Placement Resource Usage



Step 3: SUD with 2 Threads on same Cache on Quad Core

This test uses the same number of MPI ranks and the same process placement as in step 2. The only difference is that now the second core is activated by a second thread per MPI process. This allows us to precisely quantify the benefit of the second core without any noise by other effects.

The ratio between the previous (Step 2) and this timing quantifies the gain factor that can be attributed to SUD parallelism.

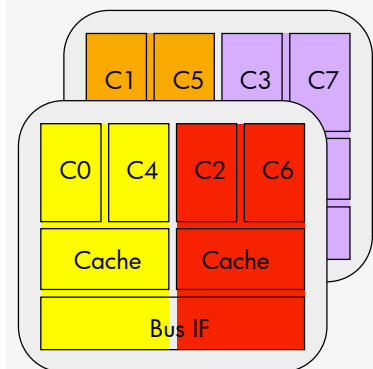
The ratio between the base timing and this timing quantifies the overall gain factor.

HP-MPI Settings:

```
-np 4 -cpu_bind=v,mask_cpu:11,22,44,88
```

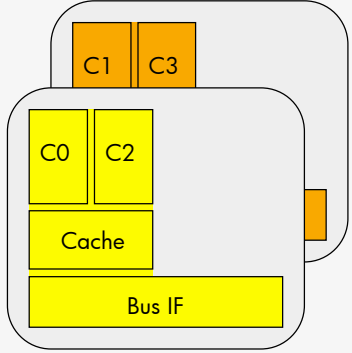
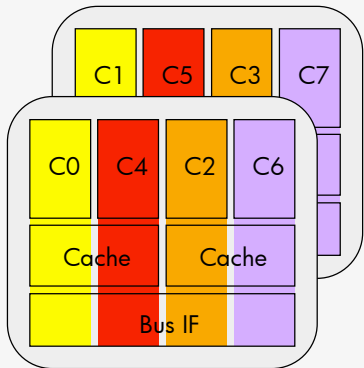
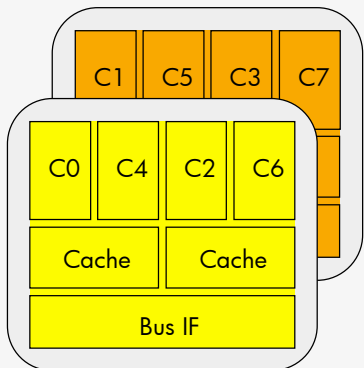
```
-e MPI_THREAD_AFFINITY=packed
```

Process Placement Resource Usage



Supplementary Measurements on Intel based nodes

The following tests were done to characterize SUD on DC nodes, to compare best case with worst-case thread placement and to evaluate SUD with 4 Threads.

<p>SUD with 2 Threads on same Cache on DC nodes</p>	<p>Process Placement Resource Usage</p>
<p>The SUD version can also be applied on DC systems if they have shared caches.</p> <p>HP-MPI Settings:</p> <pre>-np 2 -cpu_bind=v,mask_cpu:05,0A -e MPI_THREAD_AFFINITY=packed</pre>	
<p>SUD with 2 Threads on different Caches on QC nodes</p>	<p>Process Placement Resource Usage</p>
<p>This is the worst-case placement when using the SUD version. It was expected and actually confirmed by measurements that performance is 30% worse compared to the best-case thread placement.</p> <p>This confirms that thread placement on a common cache is mandatory to get good performance with the SUD versions.</p> <p>HP-MPI Settings:</p> <pre>-np 4 -cpu_bind=v,mask_cpu:03,30,0C,C0 -e MPI_THREAD_AFFINITY=cyclic_cpu</pre>	
<p>SUD with 4 Threads on common processor on QC nodes</p>	<p>Process Placement Resource Usage</p>
<p>This is a different scenario for the SUD version. This test uses 4 threads per rank with threads placed within the same processor.</p> <p>Only in very few corner cases this mode achieves better results than with 2 threads. With LM_RAPS there is no such case.</p> <p>HP-MPI Settings:</p> <pre>-np 2 -cpu_bind=v,mask_cpu:55,AA</pre>	

PAM-Crash DMP versus SUD performance

Performance characterization

PAM-Crash is available in both SMP and DMP versions. The most widely used version is DMP which scales fairly well up to 100 processors with most production test cases made of up to 1 million elements. Scalability up to 500 processors has been demonstrated with artificially large test cases made of 10 million elements [1].

PAM-Crash is quite sensitive to processor cache size and clock rate. Its memory bandwidth requirement is moderate.

Building a SUD version

Build environment

PAM-Crash was built on a Linux platform using the Intel v10.0 Compiler with its built-in OpenMP capability. HP-MPI was used for the DMP part.

Considerations around MPI thread-safety

HP-MPI comes with a thread-safe version of its libraries. Having a close look at the application it turned out that this is not needed. Thread-safe MPI is only required if several threads make concurrent MPI calls. This is not the case.

Source code modifications: None

PAM-Crash is almost ready for production of an official SUD version. Since years the SMP and DMP versions are built from the same source base by selection of preprocessor switches. Experiments with a SUD version have been made earlier at ESI-Group and other hardware vendors [1].

PAM-Crash was built with both SMP and DMP preprocessor switches turned on. The resulting executable worked instantly. The only restriction applies to the implicit solver, which is only SMP parallel. But this is not a widely used feature so it did not inhibit any of the planned tests.

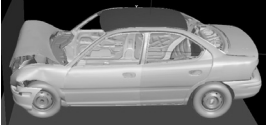
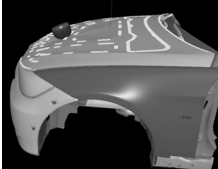
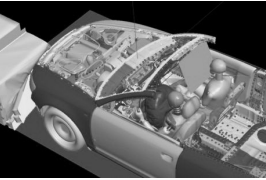
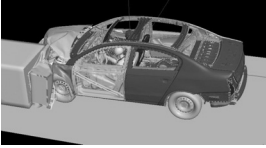
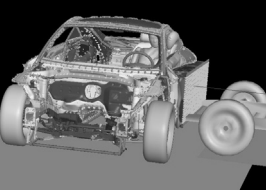

Selection of test cases

PAM-Crash data dependencies

PAM-Crash test cases can have quite different scalability characteristics. This is caused by the ratio of shell to solid elements, the wealth of material models, the use of specific submodels like dummies, airbags and barriers. The change of geometry during the simulation also has an impact on performance. For these reasons PAM-Crash was tested with 6 different test cases, most of them are real customer data sets.

Description of the PAM-Crash test cases

Table 1. PAM-Crash test cases

Name		Number of Elements:	Full simulation time	Shortened simulation time
Description		Shells Solids	Number of time steps	
Advanced Features				
Neon 100% Frontal Crash		274000 3000	120 ms 99900	n/a
none				
Q2 Pedestrian Impact		632000 22500	30 ms 42800	n/a
Dummy Head				
SMD Offset Deformable Barrier		660000 41000	140 ms 144000	n/a
Dummies, Airbags, Belts				
PST Offset Deformable Barrier		768000 40000	100 ms 100044	n/a
Dummies, Belts				
BMI Side Impact		697000 73000	90 ms 100000	30 ms
Dummies, Airbags, Belts				
Q5 Side Impact		1592000 140000	130 ms 134000	20 ms
Dummies, Airbags, Belts				

Time linearity

Each test case was first tested for its time linearity. That property describes the possibility of drawing meaningful conclusions from a shortened simulation. If time linearity is good this means that the dynamic behavior in terms of performance and load balance does not change much during a full test. That allows us to run tests with shortened simulation time. Just for verification a few runs are done with full simulation time.

Neon Measurements

Elapsed times and SUD performance gain

Figure 6. Neon elapsed times

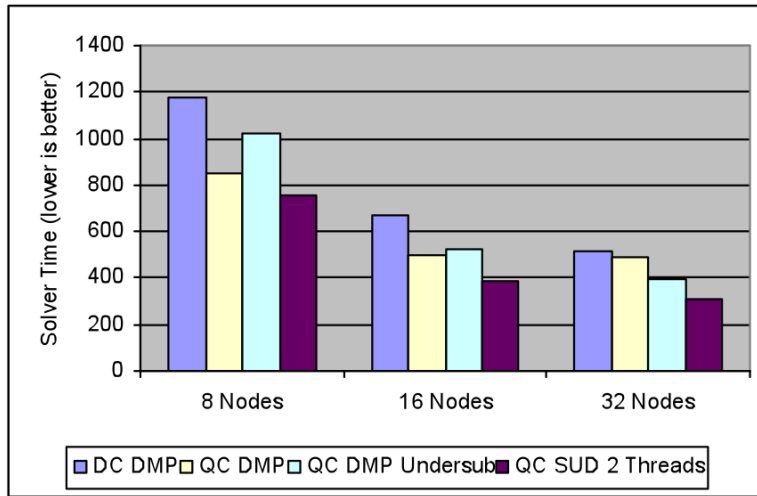


Table 2. Neon Performance gains over DC DMP

	8 Nodes	16 Nodes	32 Nodes
Additional Cores and Cache	1,39	1,33	1,05
Undersubscription	0,83	0,96	1,24
SUD Parallelism	1,36	1,36	1,26
Overall gain	1,57	1,73	1,64

Messaging analysis

Figure 7. Neon Messaging Summary

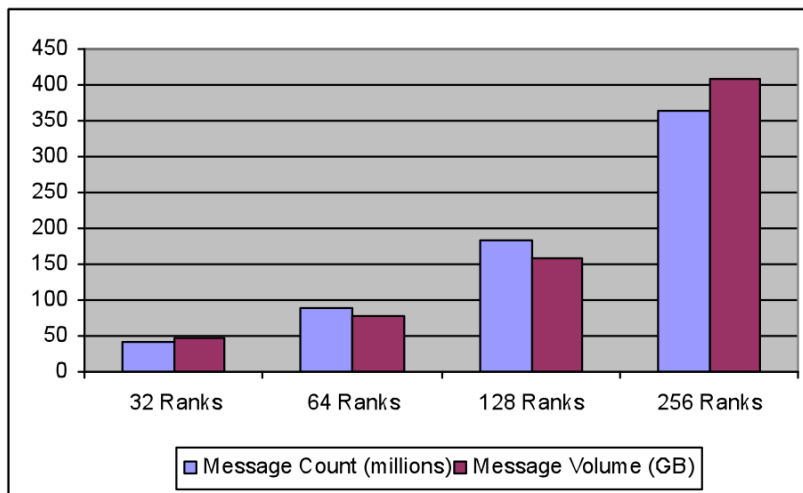


Table 3. Neon Messaging Summary

	32 Ranks	64 Ranks	128 Ranks	256 Ranks
Message Count (millions)	43	89	184	363
Message Volume (GB)	46	78	158	408

Observations

- The message count correlates linearly to the number of ranks.
- The total message volume correlates linearly to the number of ranks up to 128 ranks. Above that threshold the increase becomes superlinear.
- SUD extends scalability beyond 16 nodes by reducing the number of ranks down to a more efficient range.

Q2 Measurements

Elapsed times and SUD performance gain

Figure 8. Q2 full simulation timings

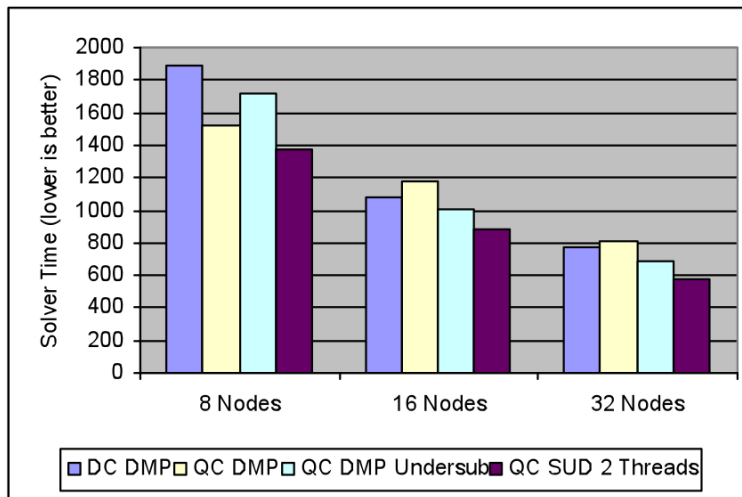


Table 4. Performance gains over DC DMP with Q2

	8 Nodes	16 Nodes	32 Nodes
Additional Cores and Cache	1,23	0,91	0,96
Undersubscription	0,89	1,18	1,19
SUD Parallelism	1,26	1,14	1,19
Overall gain	1,37	1,23	1,36

Messaging Analysis

Figure 9. Q2 Messaging Summary

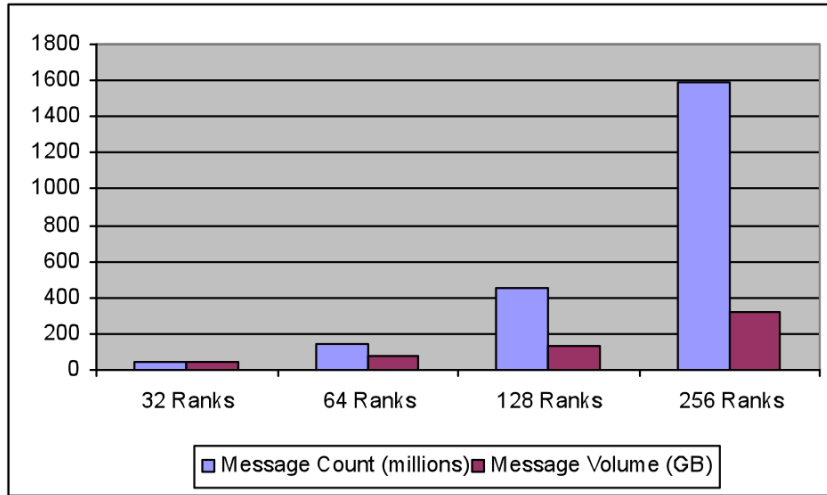


Table 5. Q2 Messaging Summary

	32 Ranks	64 Ranks	128 Ranks	256 Ranks
Message Count (millions)	47	141	452	1588
Message Volume (GB)	41	72	137	319

Observations

- The message count has an exponential correlation to the number of ranks. It is approximately an $O(1.5^n)$ function up to 128 ranks. Above that threshold it becomes worse.
- The total message volume correlates linearly all the way up to 256 ranks.
- Undersubscription and SUD improve scalability for large node counts by reducing the number of ranks down to a more efficient range.

SMD Measurements

Elapsed times and SUD performance gain

Figure 10. SMD full simulation timings

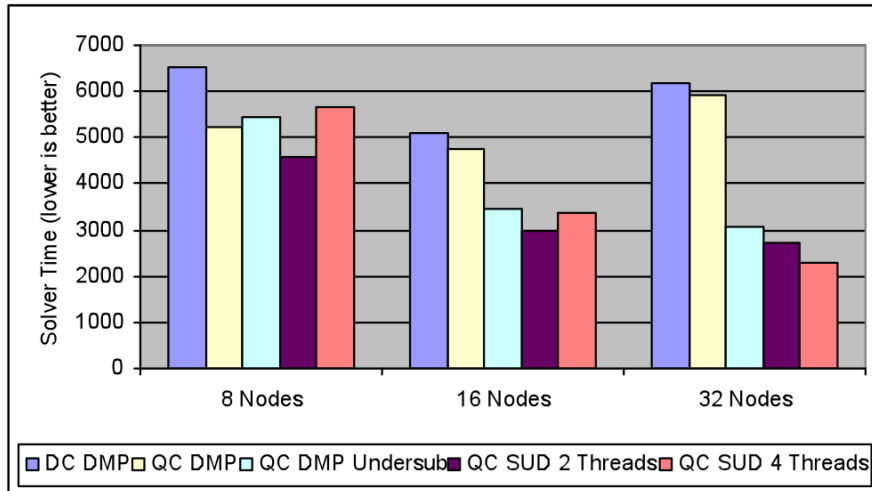


Table 6. Performance gains over DC DMP with SMD

	8 Nodes	16 Nodes	32 Nodes
Additional Cores and Cache	1,25	1,08	1,05
Undersubscription	0,96	1,37	1,93
SUD Parallelism	1,19	1,17	1,13
Overall gain	1,42	1,72	2,28

Messaging Analysis

Figure 11. SMD Messaging Analysis

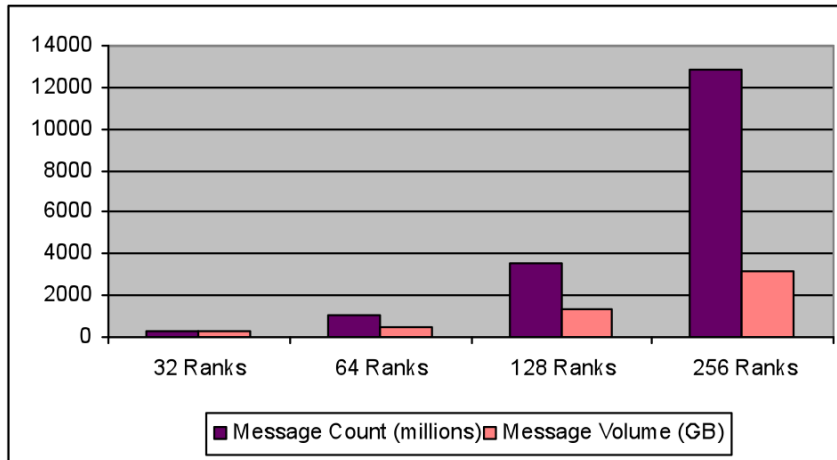


Table 7. SMD Messaging Analysis

	32 Ranks	64 Ranks	128 Ranks	256 Ranks
Message Count (millions)	298	1010	3518	12873
Message Volume (GB)	274	495	1364	3187

Observations

- The message count has an exponential correlation to the number of ranks. It is approximately a $O(3^n)$ function which is among the worst of all test cases.
- The total message volume has a roughly linear correlation only up to 64 ranks. Above that it increases exponentially just like the message count.
- Undersubscription and SUD yield enormous improvements by reducing the number of ranks down to a more efficient range.
- SUD with 4 threads is doing exceptionally well for this case. Dividing the number of ranks by 4 reduces the message count and volume by up to a factor of 10.
- This test case is a corner case with extreme behavior. It must not be used for drawing general conclusions.

PST Measurements

Elapsed times and SUD performance gain

Figure 12. PST timings

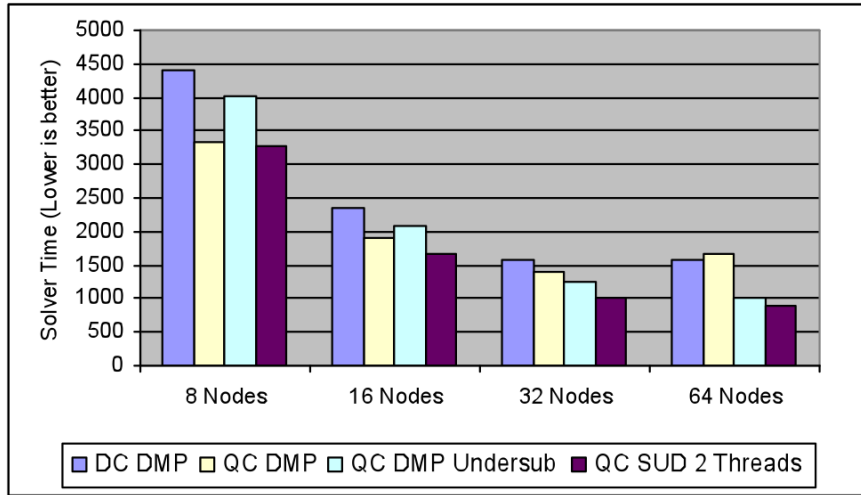


Table 8. Performance gains over DC DMP with PST

	8 Nodes	16 Nodes	32 Nodes	64 Nodes
Additional Cores and Cache	1,32	1,23	1,12	0,95
Undersubscription	0,83	0,91	1,12	1,66
SUD Parallelism	1,22	1,25	1,23	1,14
Overall gain	1,35	1,39	1,54	1,80

Messaging analysis

Figure 13. PST Messaging analysis

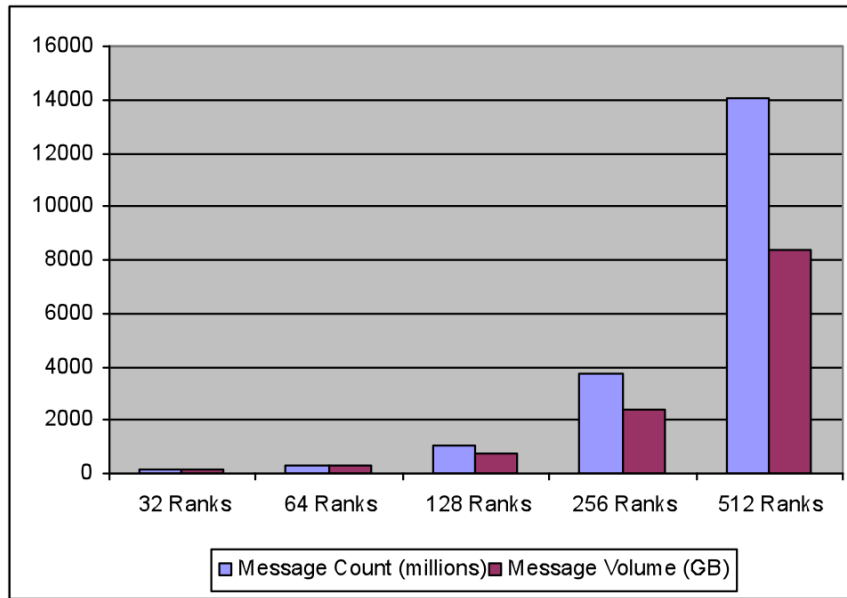


Table 9. PST Messaging Analysis

	32 Ranks	64 Ranks	128 Ranks	256 Ranks	512 Ranks
Message Count (millions)	115	327	1053	3766	14075
Message Volume (GB)	151	306	752	2356	8373

Observations

- Up to 64 ranks the message count and volume grow linearly with the number of ranks. Above that threshold the correlation becomes exponential.
- Undersubscription and SUD yield significant improvements above 16 nodes by reducing the number of ranks down to a more efficient range.
- The scalability limit is extended from 32 to 64 nodes.

BMI Measurements

Elapsed times and SUD performance gain

Figure 14. BMI timings

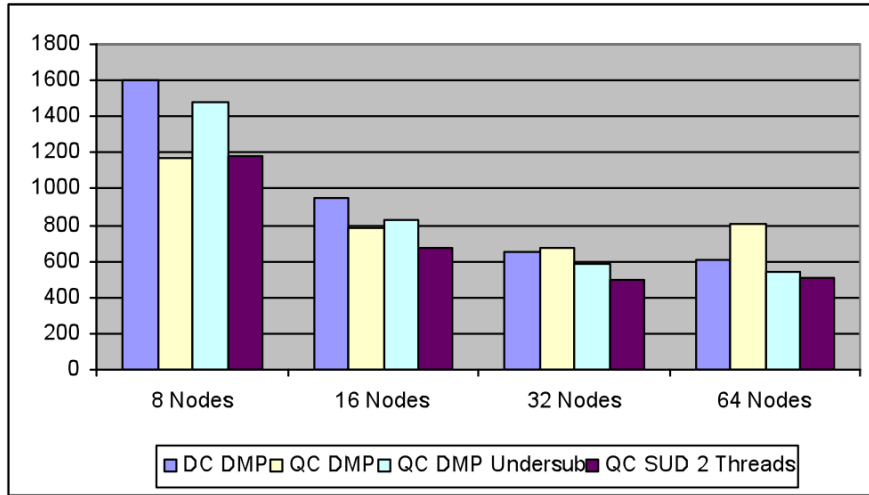


Table 10. Performance gains over DC DMP with BMI

	8 Nodes	16 Nodes	32 Nodes	32 Nodes
Additional Cores and Cache	1,37	1,22	0,97	0,75
Undersubscription	0,79	0,94	1,15	1,49
SUD Parallelism	1,25	1,24	1,17	1,08
Overall gain	1,36	1,41	1,29	1,21

Messaging analysis

Figure 15. BMI Messaging analysis

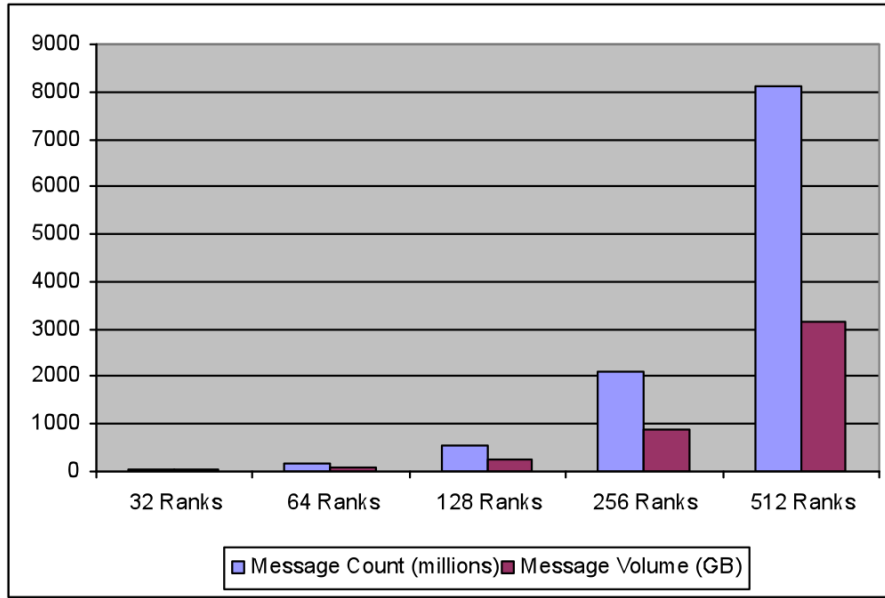


Table 11. BMI Messaging analysis

	32 Ranks	64 Ranks	128 Ranks	256 Ranks	512 Ranks
Message Count (millions)	45	153	553	2094	8130
Message Volume (GB)	53	100	269	867	3153

Observations

- Both the message count and the volume have an exponential correlation to the number of ranks. It is approximately a $O(3^n)$ function which is among the worst of all test cases.
- SUD yields substantial improvements by reducing the number of ranks down to a more efficient range. While some overhead is cut off the scalability limit of 32 nodes cannot be overcome.
- Undersubscription only makes sense with 32 and more nodes.

Q5 Measurements

Elapsed times and SUD performance gain

Figure 16. Q5 timings

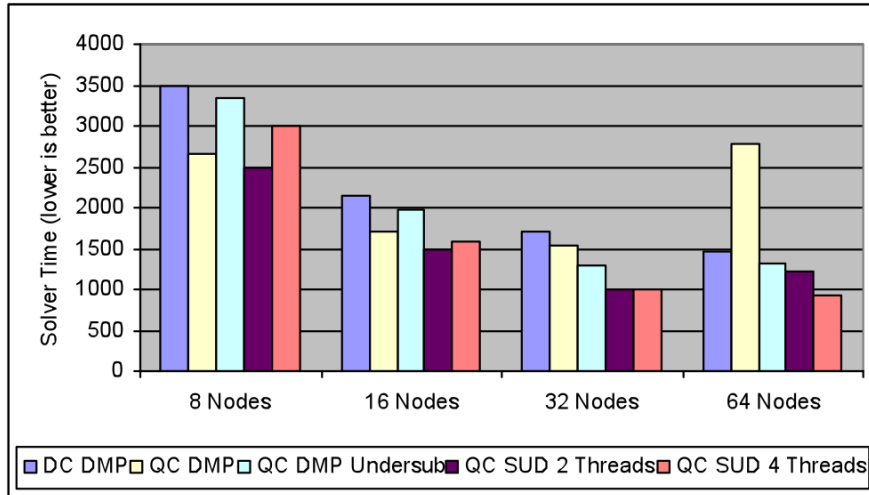


Table 12. Performance gains over DC DMP with Q5

	8 Nodes	16 Nodes	32 Nodes	64 Nodes
Additional Cores and Cache	1,32	1,26	1,11	0,53
Undersubscription	0,79	0,86	1,20	2,12
SUD Parallelism (2 or 4 Threads)	1,33	1,34	1,27	1,41
Overall gain	1,39	1,45	1,70	1,58

Messaging analysis

Figure 17. Q5 Messaging analysis

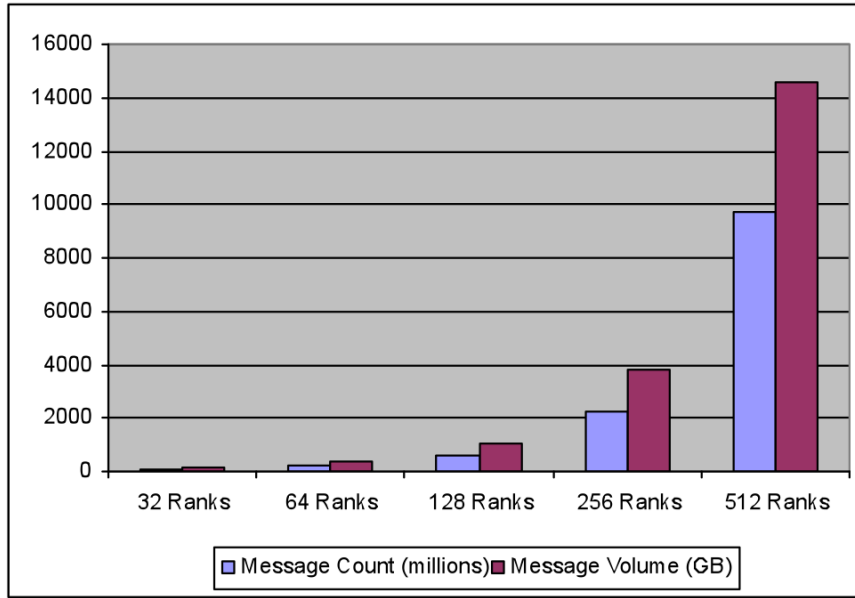


Table 13. Q5 Messaging analysis

	32 Ranks	64 Ranks	128 Ranks	256 Ranks	512 Ranks
Message Count (millions)	54	193	632	2216	9694
Message Volume (GB)	139	355	1073	3807	14577

Observations

- Both the message count and the volume have an exponential correlation to the number of ranks. It is roughly a $O(3^n)$ function which is among the worst of all test cases.
- SUD yields substantial improvements by reducing the number of ranks down to a more efficient range. Scalability for QC systems is extended from 16 to 32 nodes. SUD with 4 threads even gets a bit better with 64 nodes.
- Undersubscription only makes sense with 32 and more nodes.

Testing SUD on Dual-Core nodes

Measurements

Figure 18. Neon and Q2 on DC nodes

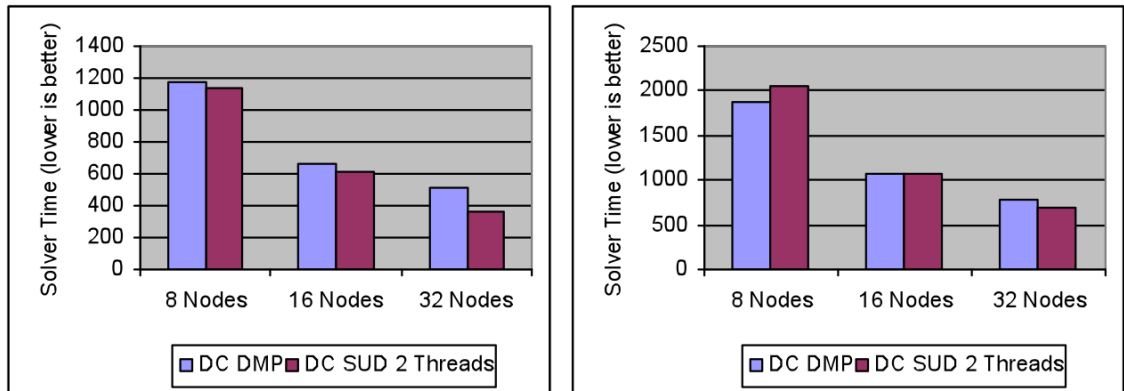


Figure 19. SAMD and PST on DC nodes

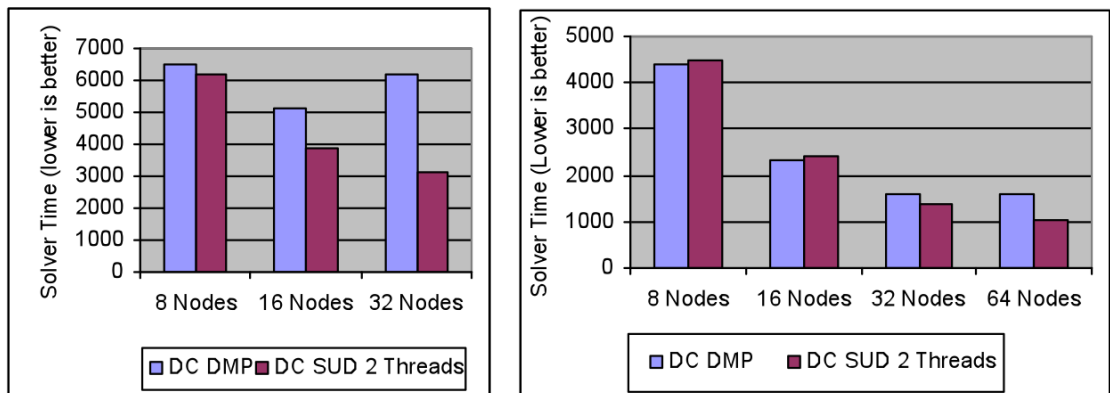
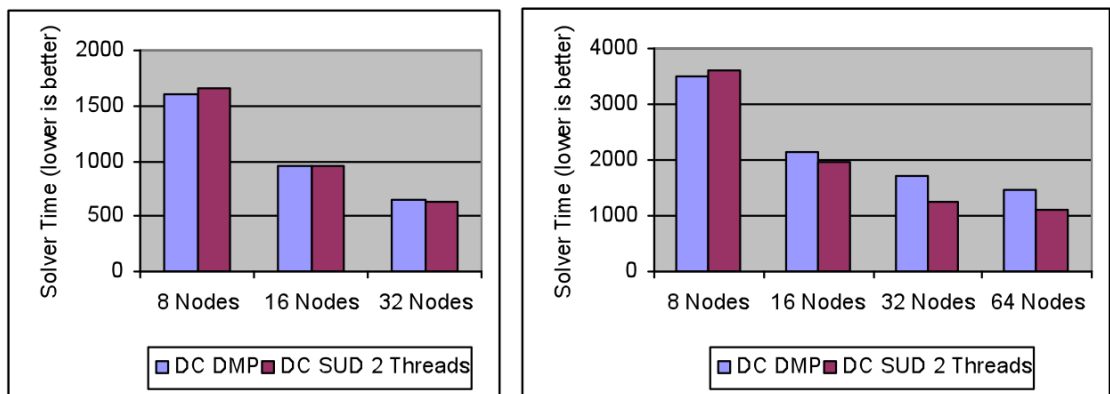


Figure 20. BMI and Q5 on DC nodes



Observations

SUD can be used on DC nodes too. At low node counts there might be a slight slowdown in some cases. At high node counts there is always a clear advantage. It is not as clear as with QC nodes but it is still visible.

With this observation the SUD version can be considered as a general solution for PAM-Crash on Multi-Core processors.

Summary of PAM-Crash performance on Intel processors

The SUD version is a valid approach to extend scalability of PAM-Crash by up to a factor of 2. This improvement is accomplished by a combination of effects:

1. Some algorithms in PAM-Crash have an exponential increase in MPI overhead with large rank counts. Reducing the rank count with SUD gets them into a more efficient mode of operation. This is the main performance improvement by the SUD version.
2. Placing 2 corresponding threads on the same shared cache is mandatory to achieve good performance. This placement increases the efficiency of the cache and minimizes coherency overhead.
3. Some cases can make good use of SUD with 4 threads for large node counts. This mode should be even better on AMD Barcelona processors, which have the L2 cache, shared between 4 cores. The test cases can be identified by their high percentage of "CONTACTS" time in the PAM-Crash internal time breakdown.
4. In many cases the SUD version yields significant performance gains even for DC processors.
5. Using the SUD version with 8 and less nodes does not yield any benefit but it does not hurt either. That makes it a viable general solution.

Even if the SUD version should not become generally available then the DMP version in undersubscribed mode is an alternative. It is not always beneficial so it has to be applied selectively.

Appendix

References

- [1] Feyereisen, M.: Presentation on PAM-Crash SUD performance at 2007 Detroit CAE Symposium
- [2] Riedmann, M., 2008: "Applying SUD Parallelism to improve parallel Efficiency of COSMO/DWD LM_RAPS 4.0 on Multi-Core Processors"

Credits

Thanks to Laurent Duhem (Intel) and Thorsten Queckbörner (ESI-Group) for their valuable application support and their suggestions during this investigation.

For more information

PAM-Crash

www.esi-group.com/SimulationSoftware/NumericalSimulation/index.html

HP High Performance Computing

www.hp.com/go/hptc

HP-MPI

www.hp.com/go/mpi

Intel cpuintfo as part of Intel MPI

<http://www.intel.com/cd/software/products/asm-na/eng/308295.htm>

© Copyright 2008 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

Linux is a U.S. registered trademark of Linus Torvalds. Microsoft and Windows are U.S. registered trademarks of Microsoft Corporation. UNIX is a registered trademark of The Open Group.

4AA1-xxxxENW, May 2008

