Efficiency analysis and parallelization of the CNSPACK implementation of a preconditioned CGS solver for modern multicore computer systems

Oleg A. Bessonov, Alexander I. Fedoseyev

Efficiency analysis and parallelization of the CNSPACK implementation of a preconditioned CGS solver for modern multicore computer systems

Oleg A. Bessonov[1], Alexander I. Fedoseyev[2*]

[1] Institute for Problems in Mechanics, Russian Academy of Scinces, Moscow, Russia
[2] Untraquantum LLC, Madison, Alabama, USA

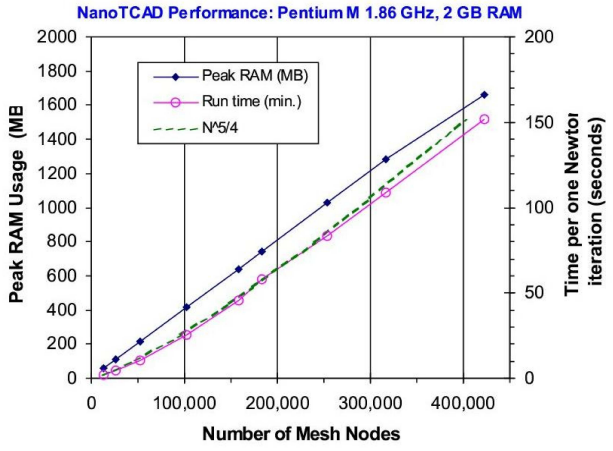
The goal of this work is the analysis of optimization and parallelization opportunities of the large sparse matrix solver CNSPACK for modern multicore microprocessors. CNSPACK is an advanced solver used for coupled solution of stiff problems [1,4]. It employs iterative CGS algorithm with ILU preconditioning (user chosen ILU preconditioning order). CNSPACK has been successfully used during last decade for solving problems in several application areas, including fluid dynamics and semiconductor device simulation. Originally, CNSPACK was implemented and optimized for early sequential processors, keeping in mind their arithmetic and memory-size limitations. In the beginning of 90's, the first optimization exercise was performed in order to accelerate (6X) the algorithm for appearing superscalar microprocessors (Intel i860) [1] (see Figure 1 for CNSPACK performance with almost linear CPU time & memory versus N).

However, there was a huge change in processor architectures and computer system organization since that time. Now, desktop computers and cluster nodes use high performance pipelined superscalar multicore processors with out-of-order execution of instructions, deep cache hierarchies and high-throughput memory capabilities. Due to this, performance criteria and methods have been revisited, together with consideration of parallelization. The latter becomes important because now parallelization is no more an option (which requires an access to expensive parallel computer systems). Now it is a must because there is no other way to fully utilize the computational potential of a processor. On the other hand, a multicore processor is a shared memory system (rather than a distributed memory cluster). This property simplifies parallelization because a convenient and natural OpenMP environment can be used [2]. This model, of course, can be extended with the distributed memory environment (MPI) for the further parallelization in the context of a cluster built upon several shared memory nodes.

It is initially known that the CNSPACK algorithm requires both the speed of arithmetic calculations and the throughput of memory accesses. Because of the computation-memory disbalance of modern processors, the latter property becomes even more important (i.e. the algorithm is memory-bound rather than computational-bound). The above determines the choice of a computer system used for the investigation. On the other hand, there are no more memory-size limitations, and algorithms can be now re-arranged correspondingly in favor of the computational speed.

In this paper we will describe the results of successful parallelization of CNSPACK including the evaluation of the computer system performance in the context of the considered class of algorithms, and analysis of optimization opportunities for serial implementations of CNSPACK:

(i) Analysis of alternative storage schemes and other improvements. Application of row-wise storage scheme for preconditioning was analyzed as a preparation step to the implementation of the block-pipelined method. Also, matrices from several application areas have been tested, and parallelization results analysis is presented.



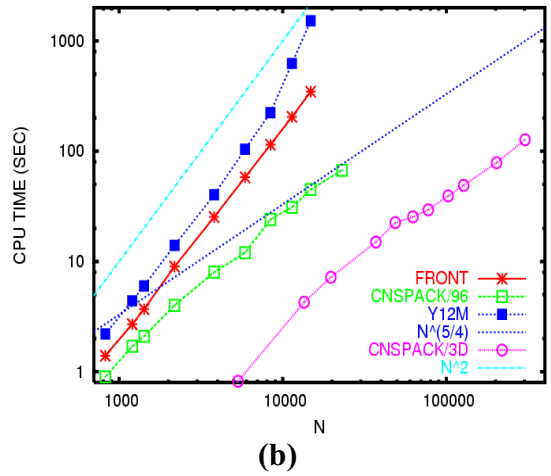


Figure 1.(a) Performance of CNSPACK linear solver for electronic device simulator in modeling 3D devices. Memory (in secs) is shown by solid curve with diamonds (in MB). CPU time is shown by solid curve with circles (per Newton iteration, (Fedoseyev 2007). One can see that the dependence on mesh nodes is almost linear for both memory and CPU time, and close to optimal theoretical estimate for CPU time, $T=O(N^{5/4})$ (dashed dark blue curve); (b) Comparison of CNSPACK software (pink color curve with circles labeled CNSPACK/3D) to other linear solvers for CFD unstructured matrices shows one to two order faster performance of CNSPACK on an Intel P6 200MHz CPU, (Fedoseyev 2001).

(ii) Implementation of advanced block-pipelined parallelization method for preconditioning. New advanced block-pipelined parallelization method of ILU-preconditioning was developed and implemented. The idea of this method is to split a matrix half-band into pairs of adjacent trapezoidal blocks that have no mutual data dependences and can therefore be processed in parallel. As a result, parallelization of the preconditioning routine for 4 threads was implemented. For this method, new blocked storage scheme was designed.

(iii) Evaluation and comparison of parallel performance of new algorithms. Performance of the new algorithm was evaluated on the target computer system (Intel Core i7-920) using two test matrices. Parallel acceleration results are presented for 4 threads. These lie within the theoretically estimated range. On a two-processor shared memory system with individual memory controllers in each processor, the new method will gain much more owing to the doubled memory throughput limitation – expected acceleration is 3.5 or more. Parallel acceleration result of the variant with single-precision stored (32-bit) matrices is 2.78 (for the CFD matrix), or 28% higher than that of for double-precision (64-bit) matrices. In comparison with the first twisted-factorization implementation variant (on single-precision), the new method

is 30% faster. On the appropriate two-processor systems, single-precision variant is also expected to obtain additional gain (can be applied to well-conditioned matrices).

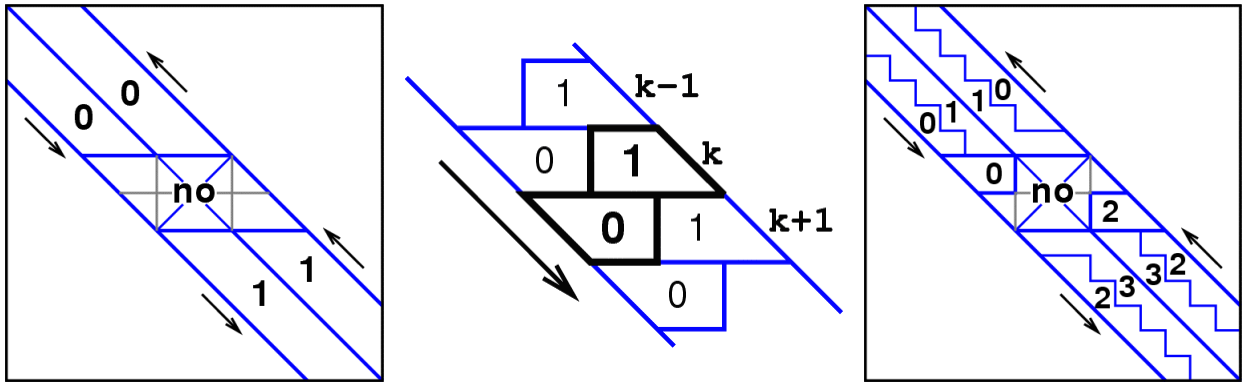


Figure 2. Illustration of implementation of pipelined parallelization of the lower part L of a matrix during the elimination process (forward sweep). Backsubstitution process (backward sweep) is implemented similarly.

Overall the CNSPACK parallelization results have reached or exceeded the expectations, making dramatic improvement to CNSPACK solver performances both in CPU and memory (both ~O(N)) with efficient use of modern modern multi-core processor capabilities. The obtained results confirmed the right choice of target computer system (Intel Core i7-9xx or, for the future, two-processor Xeon 55xx/56xx).

References

[1] A. Fedoseyev, O. Bessonov. Iterative solution of large linear systems for unstructured meshes with preconditioning by high order incomplete decomposition. Computational Fluid Dynamics Journal, V.10, No.3, 2001, 299-303. **http://www.ipmnet.ru/~bess/bess-cfdj.pdf**

[2] G. Accary, O. Bessonov, D. Fougere, S. Meradji, D. Morvan. Optimized parallel approach for 3D modelling of forest fire behaviour. Lecture Notes in Computer Science, 4671, 2007, pp.96-102. **http://www.ipmnet.ru/~bess/bess-pact2007.pdf**

[3] G. Accary, O. Bessonov, D. Fougere, K. Gavrilov, S. Meradji, D. Morvan. Efficient parallelization of the preconditioned Conjugate gradient method. Lecture Notes in Computer Science, 5698, 2009, pp.60-72. **http://www.ipmnet.ru/~bess/bess-pact2009.pdf**

[4] A. I. Fedoseyev, M. Turowski, L. Alles, and R. A. Weller, Accurate Numerical Models for Simulation of Radiation Events in Nano-Scale Semiconductor Devices, Math. and Computers in Simulation, 2007, doi:10.1016/j.matcom.2007.09.013.