Adaptive multistep predictor for accelerating dynamic implicit finite-element simulations

Kohei Fujita[1,2], Tsuyoshi Ichimura[1,2], Masashi Horikoshi[3], Muneo Hori[1,2], Maddegedara Lalith[1,2]
[1]Earthquake Research Institute & Department of Civil Engineering, The University of Tokyo
[2]Advanced Institute for Computational Science, RIKEN
[3]Software and Solutions Group, Intel K.K.

***Abstract*—We develop an adaptive multistep predictor for accelerating memory bandwidth-bound dynamic implicit finite-element simulations. We predict the solutions for future time steps adaptively using highly-efficient matrix-vector product kernels with multiple right-hand sides to reduce the number of iterations required in the solver. By applying the method to a conjugate gradient solver with 3 x 3 block Jacobi preconditioning, we were able to achieve a 42% speedup on a Skylake-SP Xeon Gold cluster for a typical earthquake ground motion problem. As the method enables the number of iterations, and thus the communication frequency, to be reduced, the developed solver was able to attain high size-up scalability: 80.6% up to 32,768 compute nodes on the K computer. The developed predictor can also be applied to other iterative solvers and is thus expected to be useful for wide range of dynamic implicit finite-element simulations.**

I. Introduction

Although the memory bandwidth of recent computers has increased due to new technologies (e.g., 3D stacked memory), the increase in arithmetic capability has been faster. Thus, it is difficult for data-intensive applications to take full advantage of the performance of recent computer hardware. Dynamic finite-element methods based on iterative solvers using sparse matrix vector products are one such application, as their performance is limited by memory bandwidth. In such finite-element methods, decreasing the number of iterations required by the solver, and thus decreasing the amount of memory transfer required, is an effective way to speed up the application. Linear multistep predictors (e.g., the Adams-Bashforth method) are widely used to generate initial solutions for the solver to reduce the number of iterations required. Improving the accuracy of the predictor is expected to further reduce the number of iterations. Since finite-element applications tend to leave much of the computer's arithmetic capability unused, we aim to use this capability to improve the accuracy of the predictor.

II. Adaptive multistep predictor for accelerating dynamic implicit finite-element simulations

In dynamic finite-element methods, the right-hand side vector in the matrix equations (b_i) depends on the solution at the previous step (x_{i-1}). Thus, the standard solution method is to step sequentially in time (Algorithm 1). Instead, our idea is to approximately solve several time steps simultaneously and use the results as a predictor. The study by Liu et al. [1] is an example of such an approach. In their method, m steps $(i, i+1, ..., i+m-1)$ are solved simultaneously at step i, and the solutions are used as initial solutions for time steps $i, i+1, ..., i+m-1$ (Algorithm 2). Although an additional step is required for the predictor (line 6), this can be solved with little computational cost since a highly-efficient kernel can be used (i.e., sparse matrix vector product with multiple right-hand sides, here called the generalized SpMV or GSpMV)[1]. Thus, even with the additional predictor cost, the reduction in the number of iterations required for the solver in line 9 leads to a 30% speedup of the whole application.

Based on this approach, we develop an adaptive multistep predictor in the present study. In the method of [1], the right-hand side vectors $\bar{b}_j$ are set based on the solutions $\bar{x}$, which are predicted using a standard predictor. Thus, the approximate solution of the simultaneous implicit solver is different from the actual x. In the present study, we adaptively update $\bar{b}$ based on the partially-solved $\bar{x}$ to improve the accuracy of the predictor using the method shown in Algorithm 3. Here, the solver at line 8 is interrupted and then restarted after the right-hand side vector $\bar{b}$ has been updated based on the partially solved $\bar{x}$ (line 6–10). For the solver at line 8, we run m instances of a conjugate gradient solver with 3×3 block Jacobi preconditioners simultaneously, restarting all the instances together when the number of iterations reaches the prescribed threshold. As well as improved prediction accuracy, high peak performance is expected since all the solvers are based on GSpMV kernels.

We compare the performance with and without the developed predictor (Algorithm 1 and Algorithm 3) using an earthquake ground motion problem with 25 time steps. Here, $m = 4$ was used for the developed method to solve time steps $i, i+1, i+2$, and $i+3$ simultaneously. A conjugate gradient solver with a 3×3 block Jacobi preconditioner was used as the base solver for both algorithms. Both algorithms were implemented based on an SC15 Gordon Bell Prize Finalist solver [2], which is a conjugate gradient-based dynamic finite-element solver that has been highly tuned for SIMD-based multi-core-CPU machines. Here, the matrix-vector product kernels are changed to 3×3 Block Compressed Row Storage (BCRS)-type SpMV and GSpMV kernels. As the base code used had been highly tuned and the modified parts were also highly tuned (e.g., software prefetching was added to hide indirect access latency), we expected the change in the predictor algorithm to be reflected in the performance.

We first checked the reduction in the number of iterations by solving 25 time steps of a wave propagation problem in a small two-layer ground model discretized with second-order tetrahedral elements. The developed predictor reduced the number of iterations from 9,068 to 3,541. On a Skylake-SP Xeon Gold cluster [3], a Haswell Xeon E5 cluster [4], and the K computer, both the SpMV and GSpMV ($m = 4$)

[1]Since the sparse matrix need only be read once from memory, regardless of the number of right-hand sides, the GSpMV kernel can be computed with higher arithmetic efficiency compared with SpMV kernels.

Algorithm 1 Base algorithm

1: set $x_{-1} = 0$
2: **for** $(i = 0; i < n; i = i + 1)$ **do**
3: guess $\bar{x}_i$ using standard predictor
4: set b_i using x_{i-1}
5: solve $x_i \Leftarrow A^{-1}b_i$ using initial solution $\bar{x}_i$
6: **end for**

Algorithm 2 Algorithm from [1] (with the notation modified to match the other parts of this paper). GSpMV is used in the iterative solver to solve multiple matrix equations simultaneously at line 6.

1: set $x_{-1} = 0$
2: **for** $(i = 0; i < n; i = i + m)$ **do**
3: guess $\bar{x}_j (j = i, ..., i + m - 1)$ using standard predictor
4: set b_i using x_{i-1}
5: guess $\bar{b}_j$ using $\bar{x}_{j-1} (j = i + 1, ..., i + m - 1)$
6: approximately solve $\{\bar{x}_j \Leftarrow A^{-1}\bar{b}_j\}$ with initial solution $\bar{x}_j (j = i, ..., i + m - 1)$
7: **for** $(j = i; j < i + m; j = j + 1)$ **do**
8: set b_j according to x_{j-1}
9: solve $x_j \Leftarrow A^{-1}b_j$ using initial solution $\bar{x}_j$
10: **end for**
11: **end for**

kernels managed to almost saturate the memory throughput. Thus, the GSpMV kernel only took approximately 1.2 times as long as an SpMV kernel on these machines. Combined with the reduced number of iterations, the application speedup was 42% on the Skylake-SP Xeon Gold cluster, 40% on the Haswell Xeon E5 cluster, and 50% on the K computer (Fig. 1). Although direct comparison is difficult due to the difference in problem settings, it is clear that the proposed method leads to fewer iterations and a shorter runtime, performing at least as well as the method in [1]. The reduced number of iterations effectively reduced total communication latency, so the size-up scaling efficiency from 1,024 to 32,768 compute nodes on the K computer improved by 2.3% (from 78.3% to 80.6%) using the proposed predictor (Fig. 2). The speedup efficiency was also good, as it can be seen that the developed method was faster in all cases, from 1,024 to 16,384 compute nodes. The proposed method also reduced the number of iterations by a factor of 2.62 and achieved a 52% speedup on a practical problem involving a 10.25 km $\times$ 9.25 km $\times$ 280 m area of central Tokyo discretized using a 4 m sized mesh, computed using 2,304 nodes of the K computer.

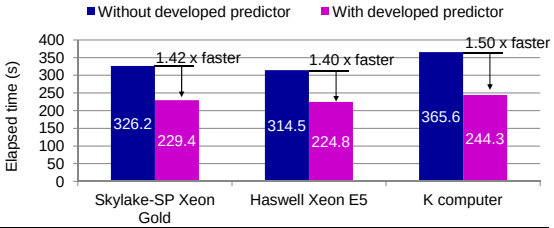
III. Summary and Future Prospects

We have developed an adaptive multistep preconditioner for accelerating dynamic implicit finite-element simulations. By adaptively reducing the number of iterations required by the iterative solver using a highly-efficient GSpMV kernel, we were able to speed up a memory bandwidth-bound application in a massively-parallel compute environment. The adaptive multistep predictor can also be applied to sophisticated iterative solvers (i.e., multi-grid iterative solvers) and is thus expected to be useful for a wide range of dynamic implicit problems.

Acknowledgments: Our results were obtained using the K computer at the RIKEN Advanced Institute for Computational Science (proposal numbers: hp160221, hp160160, hp160157, and hp170249). We acknowledge the support provided by the Japan Society for the Promotion of Science (15K18110, 26249066, 25220908, and 17K14719). We thank the Operations and Computer Technologies Division of RIKEN AICS and the High Performance Computing Infrastructure helpdesk for their support in the use of the K computer.

Algorithm 3 Adaptive multistep predictor algorithm. ϵ is the tolerance of solver. GSpMV is used in the iterative solver to solve multiple matrix equations simultaneously at line 8.

1: set $x_{-1} = 0$ and $\bar{x}_i = 0 (i = 0, ..., m - 2)$
2: **for** $(i = 0; i < n; i = i + 1)$ **do**
3: set b_i using x_{i-1}
4: $\bar{b}_i = b_i$
5: guess $\bar{x}_{i+m-1}$ using standard predictor
6: **while** $(|A\bar{x}_i - b_i|/|b_i| > \epsilon)$ **do**
7: guess $\bar{b}_j$ using $\bar{x}_{j-1} (j = i + 1, .., i + m - 1)$
8: refine solution $\{\bar{x}_j \Leftarrow A^{-1}\bar{b}_j\}$ using initial solution $\bar{x}_j (j = i, ..., i + m - 1)$
9: **end while**
10: $x_i = \bar{x}_i$
11: **end for**



	20 cores x 2 sockets x 2 nodes @ 2.4 GHz	12 cores x 2 sockets x 4 nodes @ 2.6 GHz	8 cores x 8nodes @ 2.0 GHz
Peak DP FLOPS	3072 GFLOPS	3993.6 GFLOPS	1024 GFLOPS
Peak mem. bandwidth	512 GB/s	544 GB/s	512 GB/s

Fig. 1. Performance of the developed predictor. Number of iterations is reduced from 9,068 to 3,541 by using the developed predictor.

References

[1] X. Liu, E. Chow, K. Vaidyanathan, and M. Smelyanskiy. "Improving the Performance of Dynamical Simulations Via Multiple Right-Hand Sides," IEEE 26th International Parallel and Distributed Processing Symposium, 2012.

[2] T. Ichimura, K. Fujita, P.E.B. Quinay, L. Maddegedara, M. Hori, S. Tanaka, Y. Shizawa, H. Kobayashi, and K. Minami, "Implicit nonlinear wave simulation with 1.08T DOF and 0.270T unstructured finite elements to enhance comprehensive earthquake simulation," Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, (SC'15), 2015.

[3] Intel Skylake-SP Xeon Gold series: https://newsroom.intel.com/newsroom/wp-content/uploads/sites/11/2017/07/intel-xeon-scalable-processors-overview.pdf

[4] Intel Xeon E5 v3 series: https://software.intel.com/en-us/articles/intel-xeon-processor-e5-2600-v3-product-family-technical-overview

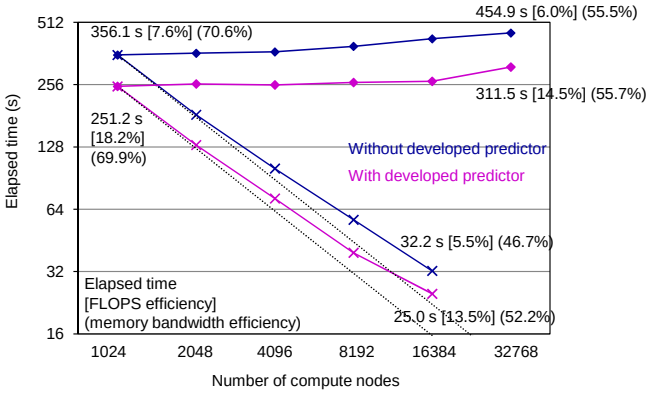


Fig. 2. Comparison of solver scalability on the K computer