PARALLEL IMPLEMENTATION OF MULTI-DIMENSIONAL ENSEMBLE EMPIRICAL MODE DECOMPOSITION

Li-Wen Chang[†]*, Men-Tzung Lo*[*]*, Nasser Anssari*[†]*, Ke-Hsin Hsu*[*]*,*
Norden E. Huang[*]*, Wen-mei W. Hwu*[†]

[†]University of Illinois at Urbana-Champaign,
Urbana, IL USA 61801
{lchang20, anssari1, w-hwu}@illinois.edu

[*]National Central University,
Chungli, Taiwan 32001
{mzlo, teethhsu, norden}@ncu.edu.tw

ABSTRACT

In this paper, we propose and evaluate two parallel implementations of Multi-dimensional Ensemble Empirical Mode Decomposition (MEEMD) for multi-core (CPU) and many-core (GPU) architectures. Relative to a sequential C implementation, our double precision GPU implementation, using the CUDA programming model, achieves up to 48.6x speedup on NVIDIA Tesla C2050. Our multi-core CPU implementation, using the OpenMP programming model, achieves up to 11.3x speedup on two octal-core Intel Xeon x7550 CPUs.

Index Terms—Multi-dimensional Ensemble Empirical Mode Decomposition, GPGPU, OpenMP, CUDA

1. INTRODUCTION

In 1990s, Huang's group at NASA developed a new adaptive time-frequency data analysis method, namely Empirical Mode Decomposition (EMD)/Hilbert Spectral Analysis [1]. In the past decade, it has been utilized in more than 3000 published works and has been applied in various research fields [2-5]. Based on one simple assumption that any time series data can be decomposed to a finite number of intrinsic modes of oscillations, EMD identifies the intrinsic undulations at different time scales and sifts the so-called intrinsic mode functions (IMFs) out. Recently, the EMD's recognized ability to suitably handle nonstationary and nonlinear signals has inspired the development of two-dimensional EMD [6-8] for the application of space/spatial-frequency representation. Nevertheless, there remains a significant issue yet to be resolved for EMD. For signals with intermittent oscillations, one intrinsic mode can comprise oscillations with very different wavelengths at different temporal locations (i.e. mode mixing), which can cause certain complications in analysis and result in less reliable conclusion. To overcome this issue, a noise-assisted EMD algorithm - Ensemble Empirical Mode Decomposition (EEMD) - has been proposed by Wu and Huang [9-10]. EEMD applies EMD to obtain an ensemble of decompositions of data with added white noise, and uses the means of the corresponding intrinsic mode functions from different decompositions as the final result. Wu and Huang also extend the concept to develop multi-dimensional EEMD (MEEMD) [11]. The ensemble approach has been well-accepted as a solution to the problem of mode mixing. However, the improved capability of MEEMD comes at dramatic increase in the computation cost which has severely limited its application.

Recently, many compute-intensive applications were shown to benefit from parallel execution on multi-core (CPU) and many-core (GPU) processors. Several programming models exist for this purpose, either targeting CPUs (e.g. OpenMP [12]), GPUs (e.g. CUDA [13]), or both (e.g. OpenCL [14]).

Several implementations of EMD, EEMD and MEEMD are available in different languages, such as R [15], Matlab, Fortran, and C [11]. On one hand, most of these implementations are sequential and thus limited in their performance growth. On the other hand, only a handful of parallel implementations are available in literature. One example is a GPU-based single-precision EMD implementation proposed by Waskito et al. [17] and reported to achieve a 27.7x speedup on an NVidia Tesla C1060 GPU over a sequential C implementation running on a 2.53GHz Intel Core 2 Duo CPU. Another example is a "GPGPU-aided" single-precision EEMD implementation proposed by Chen et al. [16] with a 31.3x speedup on an NVIDIA GTX 295 card (which contains two GPUs) over a sequential C implementation running on a 3.0GHz Intel Dual Core processor. While both implementations used an overlapped piecewise spline interpolation to approximate the original spline interpolation and get more parallelism, artifacts may arise on the boundaries of each piecewise spline interpolation.

In this paper, we propose two thread-level parallel implementations of MEEMD for multi-core CPUs and many-core GPUs. While using OpenMP and CUDA

respectively, these implementations are not limited by the underlying architecture or programming model. Our OpenMP implementation was evaluated on Intel Nehalem architecture (Xeon X7550). Our CUDA implementation was evaluated on NVidia Fermi architecture (Tesla C2050) which provides a significant improvement over previous generations in the throughput of double-precision floating-point needed in scientific analyses.

The rest of the paper is organized as follows. Section 2 introduces the MEEMD algorithm. Section 3 describes the two parallel implementations, and Section 4 discusses their performance. Section 5 concludes and describes future work.

2. PRELIMINARIES

This section introduces the MEEMD algorithm starting with its building blocks: EEMD and EMD.

Figure 1 illustrates the EMD method. For a 1D signal, the sifting procedure extracts the local extrema and traces the envelopes using cubic spline interpolation. The difference between the input and the mean of the envelopes is extracted as the output. Sifting procedures are cascaded to generate a stable residual signal. An IMF is obtained from the difference between the previous residual (the zeroth residual is defined as the source signal) and the current one. Figure 2 shows the EEMD method. An ensemble signal set is acquired by adding different independent white noises to a 1D signal and the EMD analysis is applied to each ensemble signal. The final results are the average of corresponding IMFs among the whole ensemble signal set.

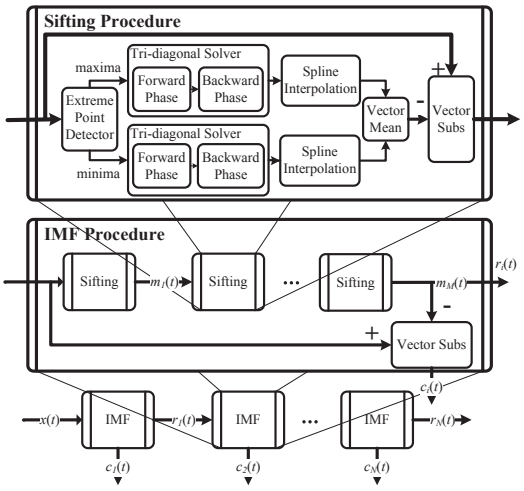

Figure 1: EMD Method

MEEMD can be explained as performing EEMD along the first dimension of a set of separate signals, then applying EEMD along the next dimension of the corresponding results from the previous EEMDs and so on. For easy understanding, the strategy of 2D MEEMD is illustrated as an example in Figure 3. After finishing EEMD executions along all dimensions, the results with the same scale are summed. In our 2D example of Figure 3, the final mode calculation would follow the following equation:

$$C_i(x,y) = \sum_{k=i}^{K} h_i^k(x,y) + \sum_{l=i+1}^{K} h_l^i(x,y)$$

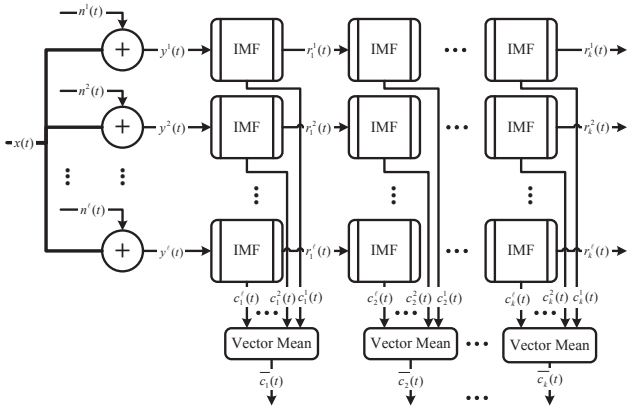

Figure 2: EEMD Method

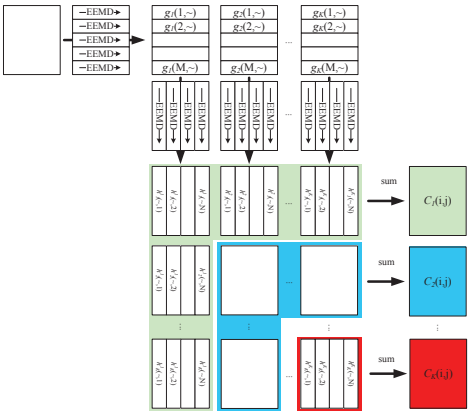

Figure 3: 2D MEEMD Illustration

3. PARALLEL IMPLEMENTATIONS

In MEEMD, although ample parallelism potentially exists in the ensemble dimension and/or the non-operating dimensions, several challenges still face a high-performance MEEMD implantation. First, **dynamic data variations**: in EEMD, white noises change the number of extrema and the sizes of the tri-diagonal linear systems, causing some irregularity and load imbalance, and thus slowing down the parallel execution. Second, **strided memory accesses of high-dimensional data**: high-dimensional data are stored in non-contiguous memory locations. Accesses along high dimensions are thus strided and uncoalesced, wasting available memory bandwidth.

Third, **limited resources to harness parallelism**: while the independent EMDs and/or EEMDs comprising an MEEMD provide high parallelism, the computational capacities of multi-core and many-core processors may not be sufficient to fully exploit the inherent parallelism of MEEMD. Moreover, increased parallelism may increase memory requirements beyond the memory capacities of these processors.

In this paper, we considered two parallel algorithm models: a thread-level model and a block-level model. Thread-level parallel algorithms use fine-grained threads in CUDA, or work items in OpenCL, to exploit massive amount of parallelism. Block-level parallel algorithms use collections of collaborative threads, referred to as thread blocks in CUDA and work group in OpenCL, to execute larger granularity tasks. In MEEMD, when a high degree of parallelism is given by the ensemble dimension and/or the non-operating dimensions, the benefits of using a thread-level parallel algorithm are threefold. First, it can exploit a higher degree of parallelism than a block-level parallel algorithm. Second, it does not incur any communication or synchronization between the threads until the final results are merged since the execution of each EMD or EEMD is independent . Finally, its implementation is similar to the sequential one, which makes it more straightforward.

3.1. OpenMP Implementation

In the CPU OpenMP implementation, the EEMDs comprising MEEMD are assigned to independent threads for parallel execution, relying on the OpenMP runtime to resolve any load imbalance issues. Strided memory accesses of high-dimensional data are eliminated by transposing these data to lower dimensions, resulting in better utilization of cache lines. The partial results of each EEMD are made thread-private for correct functionality. The required memory depends on the number of OpenMP threads and is managed by OpenMP runtime.

3.2. CUDA Implementation

In the GPU CUDA implementation, each EMD, as opposed to EEMD in the OpenMP implementation, is mapped to a thread. This allows the implementation to exploit a finer granularity of parallelism. The memory layout, especially of high-dimensional data, is rearranged to meet memory coalescing requirements and fit into the 128-byte cache lines. To this end, a corner-turning technique is applied using the low-latency on-chip memory. The data is first loaded along the lowest dimension and then consumed along a higher dimension. This step is performed when the Gaussian noise is added to form the ensemble data, and the total overhead is only less than 1% of the total execution time. In the new memory layout, the ensemble dimension is added to the lowest dimension to reduce possible branch divergence. The impact of the unavoidable branch divergence from data irregularity, caused by the noise, is minimized via a regularization technique using the on-chip memory. Moreover, the cache memory is utilized to amortize unavoidable uncoalesced memory accesses.

GPUs have a limited number of registers available to each thread, and register overuse can significantly degrade the performance. Hence, in our implementation, a GPU kernel is split into two or more kernels to minimize register pressure. The available physical memory on GPUs presents a hard limit to the problem size and parallelism degree. Our implementation inquires the available memory size on a GPU to calculate the degree of parallelism that can be supported. Based on this calculation, the non-operating dimensions can subdivided to several smaller independent MEEMDs which can be performed sequentially or scheduled to other processors.

4. RESULTS AND DISCUSSION

In our experiments, three implementations of MEEMD were compared: 1) a serial C implementation, 2) our CUDA implementation, and 3) our OpenMP implementation. CUDA experiments were executed on a desktop machine with an Intel Xeon E5520 processor and one NVIDIA Fermi Tesla C2050 GPU, with NVCC v3.1 used to compile the application. OpenMP experiments were executed on a four octal-core 2.0 GHz Intel Xeon X7550 system (totaling 32 cores, only 16 of which were used) with 64GB of memory. Intel ICC v11.0 and VTune v9.1 were used to compile and profile the application. Using an ensemble number of 1024 and a decomposition number of 5, three input data sets were used with sizes of 41×41, 333×111, and 666×444 samples.

The execution times and speedups are listed in Table 1. Our double-precision CUDA implementation achieves up to a 48.6x speedup on one GPU over the sequential C implementation. The data transfer times between the host memory and the device memory were 1.9ms, 5.8ms, and 25ms for the three input data sets respectively. Therefore, they are negligible compared to the total computation time. Contrastingly, our OpenMP implementation achieves up to an 11.3x speedup in double precision. Figure 4 shows the scaling behavior of the OpenMP implementation over 16 cores. Having a high arithmetic intensity, the application scales almost linearly over eight cores but then levels-off because of memory bandwidth limitations and Non-Uniform Memory Access (NUMA) effects on the machine.

Figure 5 shows the profiling of the different phases of the three implementations. For OpenMP, the profile is same as the one for serial C, because each OpenMP thread is essentially executing a copy of the serial code. For

CUDA, the percentage of the cubic spline interpolation decreased from 58.9% to 47.33% of the total running time because it benefits from massively parallel execution and higher memory bandwidth on GPUs. The percentage of extrema detection increased from 9.96% to 19.93% because its inherent irregular memory access pattern is inefficient on GPUs.

Table 1: Execution Time (sec) and Speedup

Size	Serial	CUDA	OpenMP (16 cores)
41×41	16.24	0.37 (43.9x)	2.13 (7.6x)
333×111	363.56	7.48 (48.6x)	32.13 (11.3x)
666×444	2929.65	62.14 (47.1x)	291.10 (10.1x)

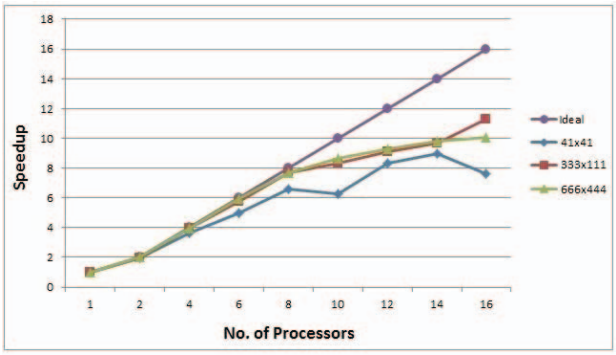

Figure 4: Scalability of Multi-core System

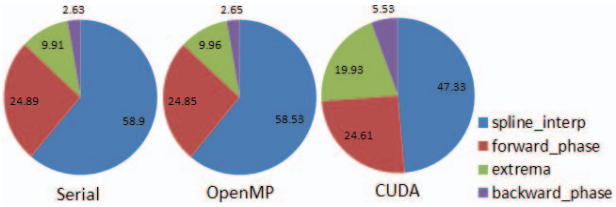

Figure 5: Profiling of MEEMD

5. CONCLUSION

In this paper, two thread-level parallel MEEMD implementations were described and shown to achieve significant speedups and good scalability on these processors compared to a sequential CPU implementation. For the current generation of multi-core CPUs and many-core GPUs, we showed that the analysis time of substantial data sets is reduced to minutes and even seconds. Experimental data show that the CUDA implementation will allow continued execution time reduction with future generation of GPUs through increased parallelism. However, several bottlenecks will need to be removed for future multi-core CPUs for similar scaling to work for the OpenMP implementation. In the future, we plan to build a library containing EMD, EEMD and MEEMD for CPUs, GPUs, and heterogeneous computing platforms.

6. REFERENCES

[1] N.E. Huang, Z. Shen, *et al.*, "The Empirical Mode Decomposition and the Hilbert Spectrum for Nonlinear and Non-Stationary Time Series Analysis," *Proceedings: Mathematical, Physical and Engineering Sciences,* vol. 454, pp. 903-995, 1998.

[2] K.T. Coughlin and K.K. Tung, "11-Year solar cycle in the stratosphere extracted by the empirical mode decomposition method," *Advances in Space Research,* vol. 34, pp. 323-329, 2004.

[3] K. Hu, M.T. Lo, *et al.*, "Nonlinear pressure-flow relationship is able to detect asymmetry of brain blood circulation associated with midline shift," *Journal of Neurotrauma,* vol. 26, p. 227, 2009.

[4] N.E. Huang, C.C. Chern, *et al.*, "A New Spectral Representation of Earthquake Data: Hilbert Spectral Analysis of Station TCU129, Chi-Chi, Taiwan, 21 September 1999," *Bulletin of The Seismological Society of America,* vol. 91, pp. 1310-1338, Oct. 2001.

[5] J.N. Yang, Y. Lei, *et al.*, "Hilbert-Huang Based Approach for Structural Damage Detection," *Journal of Engineering Mechanics,* vol. 130, pp. 85-95, 2004.

[6] C. Damerval, S. Meignen, *et al.*, "A fast algorithm for bidimensional EMD," *Signal Processing Letters, IEEE,* vol. 12, pp. 701-704, 2005.

[7] A. Linderhed, "Variable sampling of the empirical mode decomposition of two-dimensional signals," *International Journal of Wavelets, Multiresolution and Information Processing,* vol. 3, p. 435, 2005.

[8] J. Nunes, O. Niang, *et al.*, "Bidimensional Empirical Mode Decomposition Modified for Texture Analysis," in *Image Analysis*. vol. 2749, J. Bigun and T. Gustavsson, Eds., ed: Springer Berlin / Heidelberg, 2003, pp. 295-296.

[9] Z. Wu and N.E. Huang, "Ensemble Emprical Mode Decomposition: A Noise-Assisted Data Analysis Method," *Advances in Adaptive Data Analysis,* vol. 1, pp. 1-41, 2009.

[10] Z. Wu and N.E. Huang, "A study of the characteristics of white noise using the empirical mode decomposition method," *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences,* vol. 460, pp. 1597-1611, Jun. 2004.

[11] Z. Wu, N.E. Huang, *et al.*, "The Multi-Dimensional Ensemble Emprical Mode Decomposition Method," *Advances in Adaptive Data Analysis,* vol. 1, pp. 339-372, 2009.

[12] *OpenMP*: http://openmp.org

[13] *NVIDIA CUDA compute unified device architecture, programming guide, Version 3.1*, 2010.

[14] *OpenCL*: http://www.khronos.org/opencl/

[15] D. Kim and H.-S. Oh, "EMD: A Package for Empirical Mode Decomposition and Hilbert Spectrum," *The R Journal,* vol. 1, pp. 40-46, 2009.

[16] D. Chen, D. Li, *et al.*, "GPGPU-Aided Ensemble Empirical Mode Decomposition for EEG Analysis during Anaesthesia," *Information Technology in Biomedicine, IEEE Transactions on,* 2010, in press.

[17] P. Waskito, S. Miwa, *et al.*, "Parallelizing Hilbert-Huang Transform on GPU," the The 2nd Workshop on Ultra Performance and Dependable Acceleration Systems, Nov. 2010, in press.