Parallel Implementation of SEMPHY - a Structural EM Algorithm for Phylogenetic Reconstruction

E. Li, Z. Ouyang, X. Deng, Y. Zhang, W. Chen

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 631-638, 2006.

Parallel implementation of SEMPHY-a structural EM algorithm for phylogenetic reconstruction

Eric Li[a], Zhengqing Ouyang[a], Xi Deng[b], Yimin Zhang[a], Wenguang Chen[b]

[a]Intel China Research Center, Kexueyuan South Road #2, Haidian District, Beijing 100080, China

[b]Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

Inference of phylogenetic trees based on the Maximal Likelihood method is computational extremely intensive. In order to accelerate the computations, we present the parallel implementation of SEMPHY in this paper. SEMPHY is a program for the phylogenetic reconstruction from DNA/protein sequence data, which uses the structural expectation-maximization (SEM) algorithm, to efficiently search for maximum likelihood phylogenetic trees. It is dramatically faster than other ML methods while reserving comparable accuracy. Two different parallel paradigms, i.e., OpenMP and MPI version are developed to compare their performance on different parallel systems. Our experiments show that the OpenMP version scales well with the increase of processors on the shared memory system, and achieves better speedup than the MPI version on the cluster system.

1. Introduction

Phylogenetic tree represents the phylogenetic relationship of species by a tree where closely related species are placed in nearby branches. For DNA/protein sequences from different species, a phylogenetic relationship among them can be inferred to reflect the course of evolution. The phylogeny of genes or species is broadly used in gene family classification, species divergence time estimation and disease associated mutation identification, etc [1].

Phylogenetic tree inference is a high performance computing problem along with the exponential increasing of biological data [2]. The inference includes searching for the best tree topology and the best branch lengths representing the distance between the two neighbors. If we have n taxa, the number of possible rooted trees is $(2n-3)!/2^{(n-2)}(n-2)!$ and that of unrooted trees is $(2n-5)!/2^{(n-3)}(n-3)!$ [3]. Another intrinsic complexity is the numerous branch length probabilities for each topology. The search for the best tree is a NP-complete problem [4]. The exhaustive search for the whole tree space is prohibitive except that the number of taxa is small, say 12, which evaluates approximately 6.5×10^8 possible trees and takes approximately two hours on a Pentium III machine by a moderate fast tree building algorithm [5]. Currently many ongoing phylogeny practices involve tens to hundred of taxa which is unimaginable for exhaustive tree search in a reasonable time scale. Extremely, the Ribosomal Database Project-II consists of 108,781 16S rRNA sequences to be analyzed for phylogeny (RDP) [6], and the Tree of Life project aims to illustrate the evolutionary tree that unites all living things (ToL) [7]. The availability of large data sets poses a great challenge on how to both accurately and efficiently reconstruct phylogenetic trees.

There are many approaches to construct the phylogenetic tree, such as the Neighbor Joining, Maximum Parsimony, etc [8], where the maximal likelihood (ML) approach is one of the most accurate methods [9]. However, the ML method is limited in relative small data set for its huge computational intensity, even after incorporating heuristic search techniques. For example, fastaDNAml [10] takes roughly 9 days for a single run on a data set containing 150 DNA sequences of each length 1269 characters and typically tens to thousands of different runs are needed because of its heuristic search nature. For protein sequences, the problem of ML tree inference is even more demanding because the alphabet size of amino acids is twenty comparing to four of nucleotides.

Recently a new algorithm called SEMPHY is proposed for learning ML trees [11]. The main idea of SEMPHY is to use the structural expectation-maximization algorithm [12, 13], a variation of the EM algorithm for structure learning, to efficiently search for maximum likelihood phylogenetic trees. This algorithm is dramatically faster than other ML approaches while reserving comparable accuracy.

In this paper we present SEMPHY parallel implementation and analysis in both share and distributed memory environments. The parallel version of SEMPHY makes the algorithm more powerful for handling much more large data sets with hundreds of taxa, which covers most size range of current phylogenetic analysis. Particularly, it is the first time phylogenetic tree inference for large protein data set (up to 300 taxa) becomes realizable.

This paper is organized as follows. Section 2 describes the SEMPHY algorithm. Section 3 presents the analysis and parallel implementation of the SEMPHY algorithm in different parallel environments. Section 4 shows the experimental results. Section 5 concludes the paper.

2. Algorithm Description

The maximal likelihood (ML) method is a well-established statistical method of parameter estimation. The ML method of phylogenetic inference attempts to reconstruct a phylogenetic tree using an explicit evolution model. Mathematically speaking, given data D and a proper nucleotide (or protein) substitution model, ML methods will find a tree T, t such that $P(D|T,t)$ is maximized, where T and t represent the tree topology and the branch lengths respectively.

Exhaustive search of tree space (including the topology space and parameter space) that maximizes the log-likelihood of tree (T,t) becomes impractical when the number of taxa increases. Heuristic techniques are usually used to reduce the computation time, including the stepwise addition algorithm fastDNAml and the star-decomposition algorithm in MOLPHY [14], etc. Even using heuristic approaches, these methods still suffer from intensive computations. The structural EM algorithm used in SEMPHY efficiently solves this problem by using parameters found in current topology to help evaluate new topologies and thus improving the structure at each step until convergence. This is accomplished by a decomposition step of likelihood function.

X. Ma et al. [7] show that the expected log-likelihood for an arbitrary tree can be decomposed as the sum of local terms plus a constant:

$$Q(T,t) = \sum_{(i,j)\in T} l_{local}(E[S_{i,j}|D,T^0,t^0],t_{i,j}) + const \tag{1}$$

Where $E[S_{i,j}(a,b)]$ is the expected count of co-occurrences of the pair (a,b) in each position of (i,j):

$$E[S_{i,j}(a,b)] = E[\sum_{m=1}^{M} 1\{x_i[m]=a, x_j[m]=b\}] \tag{2}$$

Thus the Q score can be optimized by optimizing each l_{local} term separately within each iteration of the structural EM algorithm.

2.1. Structural EM algorithm

The structural EM is a variation of standard EM algorithm for learning structures. The general framework is similar to the standard EM procedure except that it optimizes not only the edge length but also the topology during each EM iteration. In the phylogenetic inference, the algorithm consists of the following steps:

E-step: Compute $E[S_{i,j}(a,b)|D,T^l,t^l]$ for all links of (i,j), and for all character states $(a,b) \in \sum$.

M-step I: Optimize link length by computing $t_{i,j}^{l+1} = argmax_t l_{local}(E[S_{i,j}|D,T^l,t^l],t)$. Compute $w_{i,j}^{l+1} = l_{local}(E[S_{i,j}|D,T^l,t^l],t^{l+1})$. Denote $W^{l+1} = (w_{i,j}^{l+1})$ to be the $2N-2$ by $2N-2$ matrix weights.

M-step II: Finding the spanning tree T_*^{l+1} which maximize $W^{l+1}(T) = \sum_{(i,j)\in T} w_{i,j}^{l+1}$. Transform the spanning tree to an equivalent bifurcating tree T^{t+1} satisfying $l(T_*^{l+1},t^{l+1}) = l(T^{l+1},t^{l+1})$.

SEMPHY repeats the above iterations until convergence to the local maximum. This algorithm can efficiently evaluate all possible trees in each iteration by finding a spanning tree first instead of directly finding a bifurcating tree which maximizes the likelihood. There is a standard algorithm [15] for finding spanning tree. For efficient computation in E-step and M-step I, dynamic programming is incorporated to calculate the expected counts $E[S_{i,j}(a,b)]$ in E-step. Each iteration of the structural EM algorithm is called "Semphy step" throughout the following paper.

2.2. The Best-branch-length (BBL) algorithm

In each iteration of the structural EM algorithm, BBL step in SEMPHY finds the best edge lengths of the tree immediately after finding a step-optimized topology. The EM algorithm of BBL consists of two sub steps which are similar to the E-step and M-step I of the structural EM algorithm except that the calculations are performed on the edges instead of all links of the tree .

3. SEMPHY Parallelization

SEMPHY has to do a number of EM iterations to construct the final phylogenetic tree, which is also time prohibitive especially for the large scale data sets. Parallelization of SEMPHY will make it more powerful to handle large data sets with hundreds of taxa, which covers most size range of current phylogenetic analysis.

Due to the intrinsic EM learning characteristics, we can only perform parallelization within each iteration of EM step, where each EM step will be further decomposed into several kernels. Therefore, application parallelization becomes a problem of how to efficiently parallelize these smaller kernels since the underlying algorithm does not allow higher level concurrency.

To compare the performance on different parallel systems, e.g., SMP (shared memory system) and Clusters, two paradigms, i.e., OpenMP programming model [16] and MPI, are used to parallel SEMPHY with different parallel environments.

3.1. Kernel identification and analysis

The schematic flowchart of SEMPHY is shown in Fig.1, where NJ tree means constructing an initial tree by the Neighbor Joining method. MST denotes finding the spanning tree and transforming it into an equivalent bifurcating tree. There are two major modules in the EM step: Semphy step and BBL step. Other modules such as NJ tree and MST take relatively less time.

When further investigating the Semphy and BBL step, we find that they can be decomposed into several even smaller kernels: calculation of the up and down message (CUD), calculation of counts for edges (CCE), and distance calculation (CD). As depicted in Fig 1, these three kernels are shared by the Semphy and BBL modules. Additionally, Semphy step has a unique kernel, calculation of counts for separate nodes (CCS). To analyze the time distribution of these kernels, the algorithmic complexities for these kernels are summarized as follows: CUD and CCE have $O(|\sum|^2 NM)$ time complexity, and the CCS and CD have $O(\sum|^2 N^2 M)$ time complexity. CCS and CD kernels

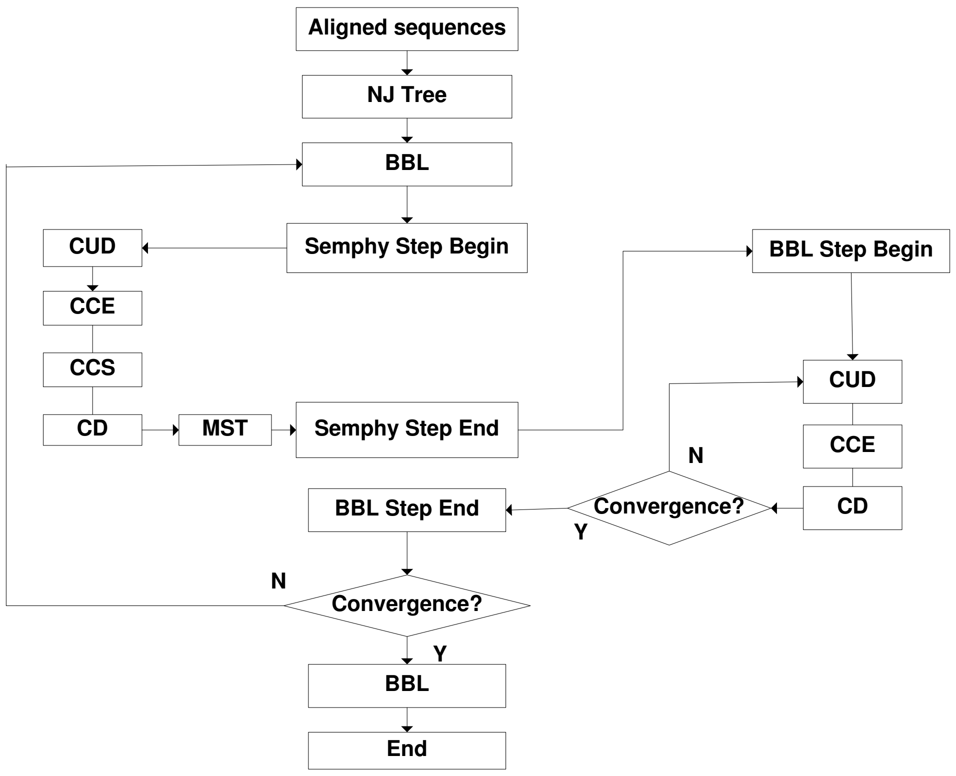


Figure 1. Flowchart of SEMPHY

will become dominating with the increase of the taxa number because the number of separate links increases in quadratic while that of edges increases only linearly. Table 1 shows the quantitative breakdown on three data sets from the Pfam database [17], which confirms the idea that the percentage of CCS and CD kernel keeps increasing with more sequences involved. Furthermore, it can be easily observed that these four kernels are the hotspots from Table 1, which occupies almost 99% of the total execution time. Parallelization can be performed on these time-consuming kernels to enhance the whole application performance.

Table 1
Kernel Breakdown for SEMPHY

Kernels	CUD	CCE	CCS	CD	Others
53 sequences	12.3%	21.88%	20.67%	44.24%	0.91%
108 sequences	8.51%	14.97%	31.0%	45.25%	0.27%
220 sequences	6.14%	9.82%	32.7%	50.6%	0.74%

3.2. Application parallelization

As aforementioned, the majority time are spent on the four kernels (CUD, CCE, CCS and CD). The parallelization of each kernel is specified below.

A. CUD

Table 2
HW/SW environments

Processor speed	3.0G	3.0G
L2 Unified Cache	512KB	512KB
L3 Unified Cache	4MB	/
L4 on-board Unified Cache	32MB	/
Interconnection	Crossbar	Gigabits Ethernet
Memory Size	8GB	1GB
Os	SUSE Linux	Redhat Linux

CUD, which computes up and down messages, including four layer nested loops. We choose the outside loop, the sequence position, to parallelize. The whole iterations are simply partitioned into N segments and they are equally distributed to the N processors. Apparently the scalability of this kernel is limited by the sequence length L and it will incur slight imbalance among all the processors when L is comparable with N.

B. CCE and CD

In SEMPHY, CCE is immediately followed by the CD kernel. Therefore they can be combined together for parallelization. Counts computations for edges are equally assigned to each processor by the edge number, and the subsequent distance computation (CD) also follows the similar decomposition strategy.

C. CCS

The process of CCS is similar to CCE. The difference is that CCE only handles all the edges, whereas CCS handles all the links in the topology tree.

In SEMPHY, the kernels for parallelization often consists of nested ”for” loops, and each item in this loop is independent with each other. Therefore the basic ”parallel for” pragma in OpenMP can be employed to handle most of these cases. Furthermore, we use dynamic scheduling policy to efficiently minimize the load imbalance impact. In spite of all of those general parallel constructs, there are still some special cases, e.g., ”while” loops and C++ iterators, and we choose an alternative Intel omp extension taskq progma to solve it, where the taskq and task pragma uses a work queuing model, serving as the extension of the OpenMP standard by Intel. Since there are few global variables and data sharings in SEMPHY, the OpenMP version can simply dispatch the computations to working threads by adding some compiler directives, without paying much attention to how to manage all of those shared variables.

Compared with OpenMP version, MPI version is considerably more difficult. It has to partition and combine the tasks by hand, carefully deal with all the communications among the working nodes. In practice, the communication cost grows dramatically with the increase of data set due to frequent collective operations, and it becomes a primary scalability limiting factor in the cluster system.

4. Results

To measure the performance of SEMPHY application, two different parallel systems are used: 16-way shared memory system and 16 node cluster. Detailed configurations of these two systems can be found in Table 2.

Table 3
Speedup performance with different dataset

	16 way SMP					
	53.phy		108.phy		220.phy	
	Time(s)	Speedup	Time(s)	Speedup	Time(s)	Speedup
1	271.24	1.0	889.9	1.0	3493.2	1.0
2	136.76	1.98	434.2	2.05	1751.5	1.99
4	71.15	3.81	222.2	4.0	873.4	4.0
8	37.83	7.17	115.3	7.72	436.7	8.0
16	23.04	11.77	62.78	14.17	224.7	15.6

	16 node cluster					
	53.phy		108.phy		220.phy	
	Time(s)	Speedup	Time(s)	Speedup	Time(s)	Speedup
1	201.47	1.0	631.1	1.0	2464.9	1.0
2	105.93	1.90	334.2	1.89	1254.5	1.96
4	58.63	3.44	186.4	3.39	663.6	3.71
8	37.25	5.41	107.9	5.85	376.7	6.54
16	29.13	6.92	72.9	8.66	234.1	10.53

For the test data, we use protein sequence data sets from the Pfam database [17], where three typical sequences: A2M_N.phy (taxa=53, length=394), AA_kinase.phy (taxa=108, length=397), and Aototransporter.phy (taxa=220, length=389) are chosen in our experiments. For simplicity, we use the taxa number to denote these sequences, e.g., 53.phy represents A2M_N protein sequence.

4.1. Scalability performance

Table 3, Fig 2,3 show the basic speedup performance for these data sets respectively. The speedup for the smaller data set, e.g. 53.phy, is linear on 2, 4 and 8 processors, but starts deteriorating when 16 processors is used on the SMP system. The slowdown on more processors occurs because the granularity of the work assigned to each processor decreases. With the increase of data set, we get much better speedup curves.

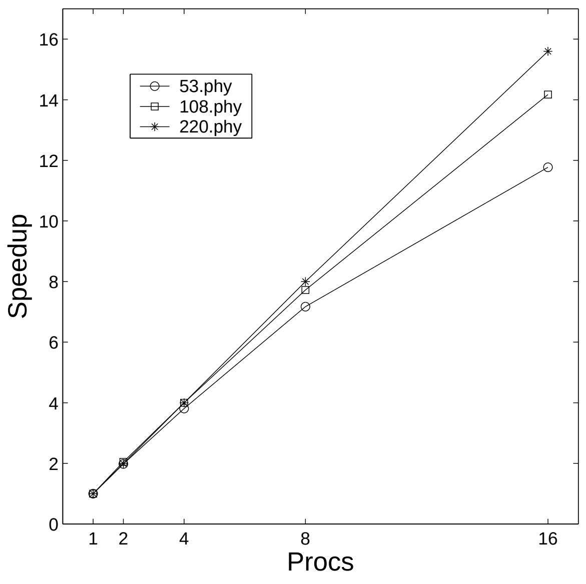
Comparing these two different parallel paradigms, the OpenMP version is consistently superior to the MPI version in terms of speedup performance. This is not surprising since the MPI version involves tremendous collective operations and the communication overhead increases with more processors.

4.2. Parallel Metrics on SMP system

To discover why SEMPHY exhibits good scalability performance on the SMP system, we further characterize its parallel performance with VTune [18] and Intel VTune Thread Profiler [18]. Table 4 and Table 5 show the metrics in detail.

In Table 4, the SEMPHY workload displays very low barrier, locks and synchronization overhead. The sequential area and the slight load imbalance may hurt the scalability performance on more processors. However, when the problem size increase, we can obtain much better performance. For example, 220.phy demonstrates a consistent better speedup curve than 108.phy on more processors.

Table 5 shows that the memory hierarchy is used efficiently for SEMPHY application, exhibiting very few cache misses, little bus conflicts and low memory bandwidth requirement, which indicates



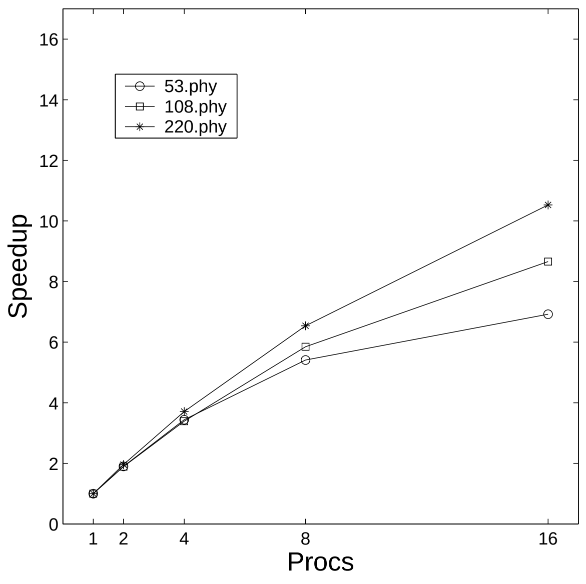


Figure 2. Speedup performance on 16-way SMP (left)

Figure 3. Speedup performance on 16 node cluster (right)

that SEMPHY can scale up well on shared memory systems.

Overall speaking, the SEMPHY application exhibits very good scalability performance in the shared memory multiprocessor system, whereas frequent collective operations and large amount of data communication inhibit its speedup on the cluster environment. It can be expected that the increasing of the hardware resource such as processor numbers on the SMP system will increase its scalability performance accordingly.

5. Conclusions

Maximum likelihood analysis is the most computationally intensive approach to phylogenetic inference. SEMPHY is dramatically faster than other ML approaches while reserving comparable accuracy. In this paper, we present a parallel implementation of SEMPHY, extracting kernels within one iteration of EM step and paralleling them with two different parallel programming paradigms. The experiments demonstrate that SEMPHY scales well on the shared memory system, achieves up to 15.6x speedup on 16 processors. However, the performance on the cluster environment is not satisfying due to huge communication cost.

References

[1] M.Holder, et al: Phylogeny estimation: traditional and Bayesian approaches. Nat. Rev. Genet., 4: 275-284. 2003.

[2] C.A.Stewart, et al: Parallel implementation and performance of fastDNAml: a program for maximum likelihood phylogenetic inference. SC2001. 2001.

[3] J.Felsenstein: The number of evolutionary trees. Syst. Zool., 27:27-33. 1978.

[4] H.L.Bodlaender, et al: Two strikes against perfect phylogeny. In Proceedings 19th International Colloquium on Automata, Languages and Programming, pp. 273-283, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1992.

Table 4
General parallel metrics with 220.phy sequence

Proc Num	Parallel Area	Sequential Area	Imbalance	Barrier	Locks	Sync.
2	99.7%	0.25%	0.04%	0.0%	0.0%	0.0%
4	99.1%	0.74%	0.17%	0.0%	0.0%	0.0%
8	97.9%	1.6%	0.5%	0.0%	0.0%	0.0%
16	95.2%	3.0%	1.8%	0.0%	0.0%	0.0%

Table 5
Memory hierarchical metrics with 220.phy sequence

Procs	L1 miss rate	L2 miss rate	L3 miss rate	Bandwidth(MB/s)	Bus Access Latency(clks)
1	2.9%	12.4%	8.2%	23.4	183.6
2	3.6%	10.3%	9.0%	50.3	190.2
4	3.5%	12.0%	7.5%	106.4	194.4
8	3.3%	10.7%	9.3%	216.7	188.2
16	3.5%	10.6%	9.3%	447.9	191.7

[5] D.L.Swofford, et al: The phylogenetic handbook, pp. 160-206, Cambridge University Press. 2004.

[6] Ribosomal Database Project-II Release 9.22. http://rdp.cme.msu.edu.

[7] Tree of Life. http://tolweb.org/tree/phylogeny.html.

[8] J. Felsenstein: Inferring Phylogenies, Sinauer Associates, Sunderland, Massachusetts. 2003.

[9] J.J.Wiens, et al: Phylogenetic analysis and intraspecific variation: performance of parsimony, likelihood, and distance methods. Syst. Biol., 47: 228-253.1998.

[10] G.J.Olsen,et al: fastDNAml: A tool for construction of phylogenetictrees of DNA sequences using maximum likelihood. Comput. Appl. Biosci. 10: 41-48. 1994.

[11] N.Friedman, et al: A structural EM algorithm for phylogentic inference. J. Comput. Biol., 9: 331-353. 2002.

[12] N. Friedman: A Structural EM algorithm for Phylogenetic Inference. Journal of Computational Biology, 9:331-353, 2002.

[13] N. Friedman. Learning belief networks in the presence of missing values and hidden variables. In Fisher, D. ed., Proceedings of the Fourteenth International Conference on Machine Learning, pp. 125-133, Morgan Kaufman, San Francisco, 1997

[14] J. Adachi, et al: Molphy version 2.3, programs for molecular phylogenetics based on maximum likelihood, Technical report, The Institute of Statistical Mathematics, Tokyo, Japan.1996.

[15] Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem, Proceedings of the American Mathematical Society, 7: 48-50, 1956.

[16] OpenMP Architecture Review Board: ”OpenMP C and C++ Application Program Interface,” Version 2.0, March 2002, http://www.openmp.org.

[17] Bateman, A., et al: The Pfam Protein Families Database. Nucleic Acids Research Database Issue 32:D138-D141, 2004.

[18] Intel Corp.: Intel Vtune Performance Analyzer, (available on-line: http://developer.intel.com/software/products/vtune/).