Editors: Volodymyr Kindratenko, kindr@ncsa.uiuc.edu
Pedro Trancoso, pedro@cs.ucy.ac.cy

From GPGPU to Many-Core: Nvidia Fermi and Intel Many Integrated Core Architecture

By Alexander Heinecke, Michael Klemm, and Hans-Joachim Bungartz

Comparing the architectures and performance levels of an Nvidia Fermi accelerator with an Intel MIC Architecture coprocessor demonstrates the benefit of the coprocessor for bringing highly parallel applications into, or even beyond, GPGPU performance regions.

In recent years, we've observed a strong trend towards using accelerators, such as GPUs and field-programmable gate arrays (FPGAs), to speed up scientific applications. In 2010, Intel announced the Intel Many Integrated Core Architecture (Intel MIC Architecture)—a general-purpose, many-core coprocessor that improves the programmability of such coprocessing devices by supporting a well-known shared-memory execution model that is based on the Intel Architecture. Here we compare the architectural features of an Nvidia Fermi accelerator and the Intel MIC Architecture and demonstrate their performance levels.

Architectural Overview

Today's scientific compute facilities need to satisfy a steadily increasing computational demand with the applications they run, while being more focused on energy consumption. For the future, experts expect heterogeneous architectures with moderate amounts of "fat" cores and a large number of accelerators or coprocessors. The November 2011 Top500 list (see www.top500.org), as well as recent announcements by the Texas Advanced Computing Center (TACC)[1] and Oak Ridge National Laboratory (ORNL),[2] show that next-generation supercomputers will use coprocessors or accelerators to speed up computation in addition to traditional CPUs.

The Intel MIC Architecture[3] coprocessor and the Nvidia Tesla C2050 accelerator[4] (which features the Nvidia Fermi architecture) are compute devices with increased compute capabilities compared to traditional (host) CPUs. The Intel MIC Architecture was announced in 2010 as a massively parallel coprocessor. It's currently available as a prerelease hardware, code-named *Knights Ferry* (based on Intel's previous Larrabee design,[5] see Figure 1). Knights Ferry and the Nvidia Tesla C2050 devices follow different design principles (see Figure 2). Whereas the Intel MIC Architecture is a many-core coprocessor based on Intel Architecture (IA), the Tesla C2050 is a massively parallel accelerator with specialized processing elements. Both devices deliver about the same peak performance and have the same power envelope.

The Intel Knights Ferry coprocessor plugs into a standard PCI Express slot. The coprocessor consists of 32 general-purpose cores running at 1,200 megahertz (MHz). The cores are based on a refreshed Intel Pentium (P54C) design and can execute 64-bit scalar instructions as well 512-bit vector instructions (16 single-precision or eight double-precision floating-point values per vector instruction). Each core can execute four hardware threads with round-robin scheduling between instruction streams, so that during each cycle the next instruction stream is selected. Knights Ferry uses the typical cache structure of per-core level-one (L1; 32-Kbyte) and level-two (L2; 256-Kbyte) caches. The shared L2 cache with a total of 8 Mbytes (32 cores) uses a high-bandwidth ring bus for fast on-chip communication. An L3 cache doesn't exist because of the high-bandwidth graphics double-data rate, version 5 memory (GDDR5; working at 125 Gbytes per second (GBps) at 1,800 MHz). Because Knights Ferry follows the key principles of the IA platform, all caches and the coprocessor memory are fully coherent.

In contrast to the Intel MIC Architecture, the Nvidia Fermi Architecture doesn't contain general-purpose compute cores. Instead, it consists of 14 multiprocessors with 32 processing elements each. The processing elements run at a clock

 1521-9615/12/$31.00 © 2012 IEEE

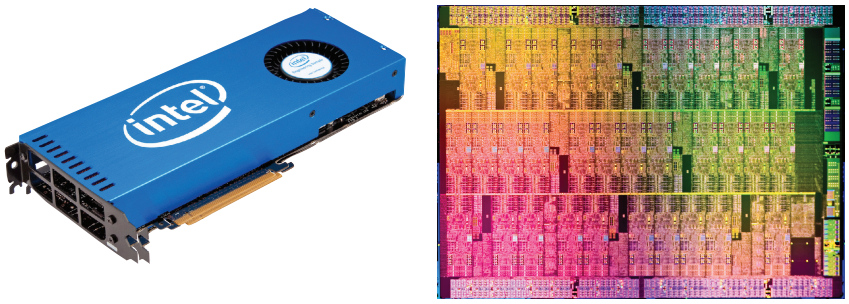

Figure 1. The Intel Many Integrated Core Architecture (Intel MIC Architecture) is currently available as prerelease hardware called Knights Ferry. (a) A Knights Ferry coprocessor board and (b) the Knights Ferry coprocessor die.

speed of 1.15 gigahertz (GHz) and a memory bandwidth of 144 GBps. A 768-Kbyte L2 cache is shared across the 14 multiprocessors. Each multiprocessor features a 64-Kbyte L1 cache. This cache can be separated into a part managed by the hardware itself (48 Kbytes/16 Kbytes) and explicitly by the programmer (16 Kbytes/48 Kbytes). Because this device is based on the Nvidia Fermi architecture, it doesn't offer vector instructions the way that the Intel MIC Architecture does; the 32 processing elements per multiprocessor are instead programmed by the so-called single-instruction, multiple threads (SIMT) paradigm. All processing elements execute either the same instruction or some execute no-operation instructions in case of conditional branches.

Based on IA, the Intel MIC Architecture supports all programming models that are available for traditional IA processors. The compilers for the Intel MIC Architecture support Fortran (including Co-array Fortran) and C/C++. We can use Open Multi-Processing (OpenMP) and Intel Threading Building Blocks for parallelization as well as emerging parallel languages such as Intel Cilk Plus. The Intel Composer XE for MIC can automatically generate vector-processing-unit (VPU) code either through autovectorization, semiautomatically by pragmas or array syntax (guided vectorization), or manually through intrinsic functions.

Because of its special architecture, the C2050 only supports a limited set of programming paradigms. The most important are CUDA and Open Computing Language (OpenCL), which are data-parallel programming models. Both offer minimal support for task parallelism, as several kernels can be invoked concurrently on recent GPU devices. Third-party compilers—such as the Portland Group, Incorporated (PGI) compiler suite or HMPP Workbench support offloading Fortran code to the GPU, but they restrict the language features in offloaded code fragments so as not to violate the GPU programming model.

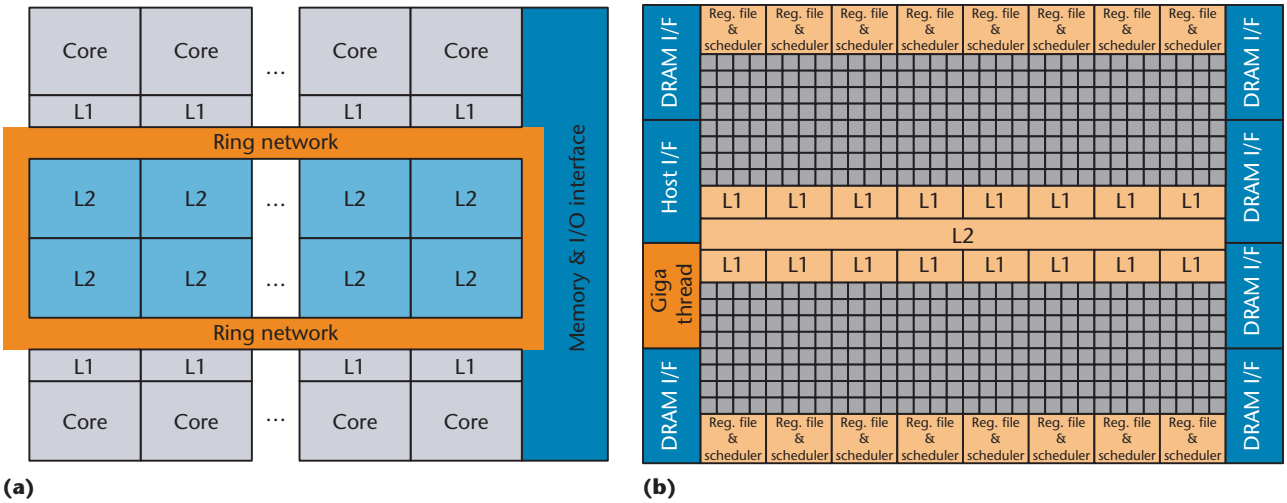

Figure 2. Comparison of design principles for a coprocessor and an accelerator. High-level view of (a) the Intel MIC Architecture and (b) the Nvidia Fermi architecture. The drawing of Fermi is based on Nvidia's diagram of the architecture.[4] (I/F stands for interface, L1 stands for level-one cache, L2 stands for level-two cache, and Reg. stands for register file.)

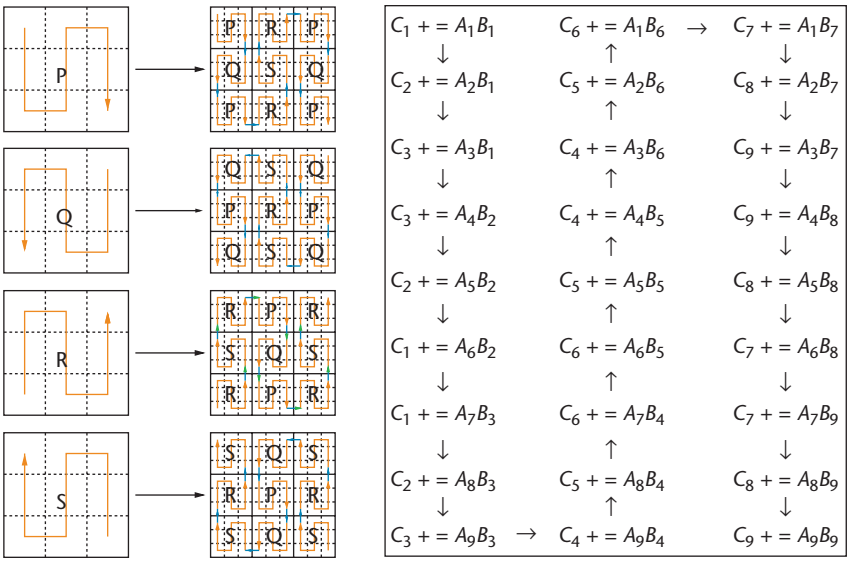

Figure 3. Processing order of submatrices in TifaMMy, a cache-oblivious implementation of matrix operations. (a) Recursive definition of matrix-storage order in TifaMMy. (b) By accessing only a preceding or subsequent matrix element, the algorithm achieves a high spatial locality.

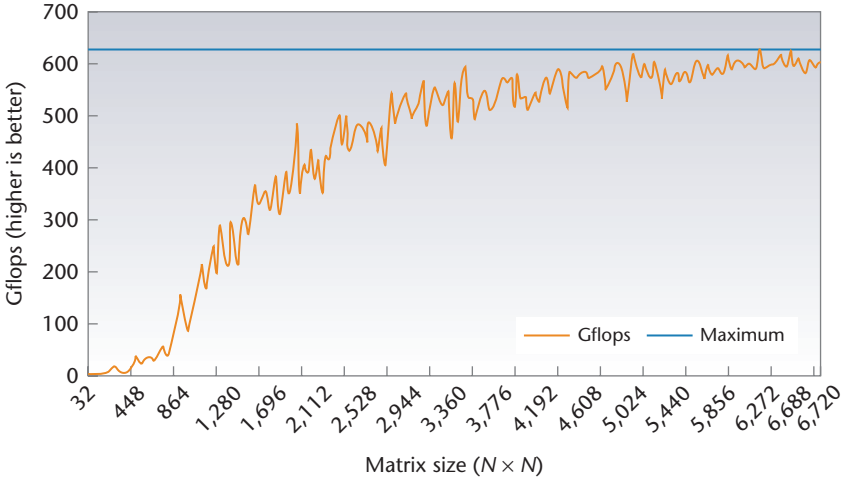

Figure 4. Performance of the TifaMMy matrix-multiplication implementation on Intel Knights Ferry. Its peak performance is around 620 gigaflops (Gflops).

Performance Evaluation

To evaluate the Knights Ferry prototype's performance, let's consider two data-parallel workloads. First, we investigate a set of self-tuning matrix operations. Then we examine a grid-based data-mining application. All performance numbers are given in gigaflops (Gflops) for single-precision floating-point numbers.

TifaMMy: Cache-Oblivious Matrix Operations

TifaMMy (http://sourceforge.net/projects/tifammy) is a cache-oblivious implementation of matrix operations. It uses space-filling Peano curves and offers an intuitive C++ API to access the algorithms. It uses a recursive scheme to partition the input data for computation and parallelization. This partitioning works well on standard CPUs (as we show elsewhere[6]). However, because of the algorithms' recursive nature, a port to any GPU-based accelerator is impracticable, as it would involve major code changes. Therefore, for this application we can only present results obtained on the Intel MIC Architecture.

Figure 3a depicts the block-recursive storage order of matrixes in TifaMMy's algorithms. The algorithms used don't require any manual tuning (for cache sizes or memory hierarchy, for example). Because only a preceding or subsequent matrix element is accessed, the implementation of TifaMMy achieves a high spatial locality (see Figure 3b). Such self-tuning algorithms effectively hide architectural changes when, for instance, running the code on the Intel MIC Architecture.

Porting the TifaMMy code to the Intel MIC Architecture was straightforward and easy to accomplish. All standard programming methods are applicable for Knights Ferry, which helps port self-adapting methods to this coprocessor. Of course, other classes of algorithms, such as partial differential equation (PDE) solvers, could benefit from the hybrid coprocessor architecture that effectively supports irregular parallelization patterns equally well.

Figure 4 plots the performance of TifaMMy's matrix-multiplication implementation on Knights Ferry. Its peak performance is roughly 620 Gflops. A two-socket system equipped with the Intel Xeon X5680 processor achieves about 230 Gflops. We want to especially highlight that the Knights Ferry prototype already outperforms the Xeon-based machine for small problem sizes.

Data Mining Based on Sparse Grids

We now turn to a workload for regression and classification algorithms

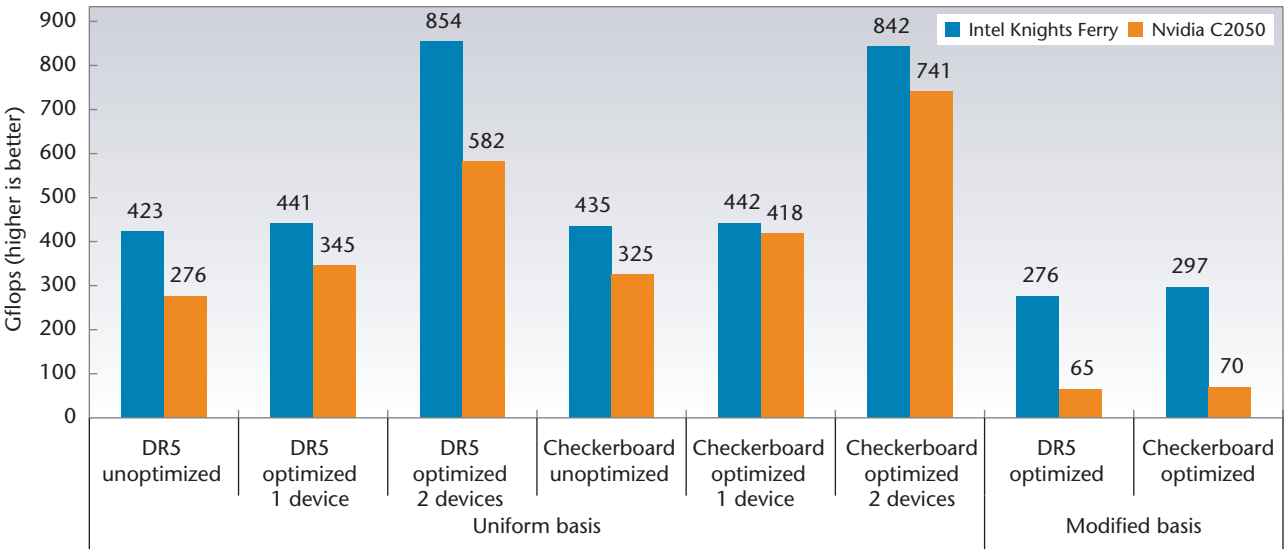

Figure 5. Performance of the data-mining application on the Intel Knights Ferry coprocessor and Nvidia C2050 accelerator. Intel Knights Ferry delivers better performance than the Nvidia Tesla C2050. DR5 stands for data release 5. We provide a detailed discussion of the application's performance elsewhere.[10]

in data-mining problems. They can be considered scattered data-approximation problems; both start from m known observations, $S = \{(\vec{x}_i, y_i) \in \mathbb{R}^d \times \mathbb{R}\}_{i=1,\ldots,m}$, with the aim to learn the functional dependency $f(\vec{x}_i) \approx y_i$ as accurately as possible. Reconstructing a smooth function f then allows an estimate $f(\vec{x})$ for new properties $\vec{x}$.

We aim at representations, $f = \sum_{j=1}^{N} \alpha_j \varphi_j(\vec{x})$ as a linear combination of N basis functions $\varphi_j(\vec{x})$ with coefficients α_j. To obtain an algorithm that scales only linearly in m, we associate the basis functions to grid points on some grid, rather than fitting their centers to the data. We rely on *adaptive sparse grids* (see elsewhere[7,8] for details) to mitigate the curse of dimensionality: regular grids with equidistant meshes and k grid points in each dimension contain k^d grid points in d dimensions. We employ two kinds of basis functions: *uniform* and *modified nonuniform*. Uniform basis functions lead to grids with a large number of grid points on the domain's boundary, whereas modified nonuniform functions extrapolate towards the domain's boundary, which lead to a smaller grid structure.

The function f should be as close to the data S as possible (minimizing the mean-squared error). At the same time, close data points should have similar function values to generalize from the data. We minimize the tradeoff between both regularization parameter λ and the hierarchical basis, allowing for a simple generalization functional:

$$\arg\min_f H[f]$$

$$H[f] = \frac{1}{m}\sum_{i=1}^{m}(f(\vec{x}_i) - y_i)^2 + \lambda \sum_{j=1}^{N} \alpha_j^2 .$$

This leads to a system of linear equations, with matrix $\boldsymbol{B}$, $\boldsymbol{B}_{i,j} = \varphi_i(\vec{x}_j)$, and identity matrix $\boldsymbol{I}$:

$$(\boldsymbol{B}\boldsymbol{B}^T + m\lambda \boldsymbol{I})\vec{\alpha} = \boldsymbol{B}\vec{y}.$$

In the following, we use two test scenarios, both with a moderate dimensionality of $d = 5$ and distinct challenges. The first dataset with 2^{18} data points classifies a regular $3\times\cdots\times3$ checkerboard pattern. The second one is a real-world dataset from astrophysics, predicting spectroscopic redshifts of galaxies based on more than 430,000 photometric measurements. For both, we obtained excellent numerical results using our method (we present the details elsewhere[9]).

Because this workload doesn't require any programming constructs like recursion and is data parallel, we were able to port it to OpenCL with moderate effort. We didn't use CUDA because OpenCL allows runtime code generation, which results in implementation and performance advantages (we provide the details elsewhere[10]).

Figure 5 compares the achieved performance on Knights Ferry with a current Nvidia Tesla accelerator. Optimizations like shared memory prefetches (or similar) are able to speed up Tesla's performance significantly, whereas explicit cache prefetches only slightly increase Knights Ferry's performance. As the MIC code was created within hours from the CPU's version, this workload clearly emphasizes Knights Ferry's ease of use. Using the nonuniform and modified grids, nested `if` statements occur in the innermost loop. Here Fermi's performance drops, while Knights Ferry is able to keep the level of high performance. The IA-based cores of Knights Ferry can execute `if` statements more effectively than the Fermi architecture. Thus, the significantly smaller number of grid points leads to a noticeable decrease

of the overall execution time. Last but not least, Intel MIC also supports multicoprocessor configurations. By using two Knights Ferry devices simultaneously, we see a speedup of 1.9×, whereas two Tesla devices only yield 1.7×.

We demonstrated that Intel MIC Architecture devices can easily be used to bring highly parallel applications into, or even beyond, GPU performance regions (data mining with adaptive sparse grids). These devices can even be used for applications that aren't feasible on GPUs (such as TifaMMy). Using well-known programming models such as OpenMP and vectorization, the Intel MIC Architecture minimizes the porting effort for existing high-efficiency processor implementations. Moreover, programming on the Intel MIC Architecture doesn't require any special tools, because its support is integrated into the complete Intel tool chain, ranging from compilers over math libraries to performance analysis tools. As future HPC systems will most likely be hybrid machines with fat cores and coprocessors, programming for the Intel MIC Architecture eases the burden for developers; codes developed for the system's CPU portion can be reused on the coprocessor without too much of a porting effort.

Acknowledgments

Intel, Pentium, Xeon, and Cilk are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. Other brands and names are the property of their respective owners.

For the Intel Knights Ferry coprocessor and Nvidia C2050 accelerator in Figures 4 and 5, performance tests are measured using specific computer systems, components, software, operations, and functions. Any change to any of those factors might cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. System configuration: Intel Shady Cove Platform with 2S Intel Xeon processor X5680 (24-Gbyte DDR3 1333, SLES11.1) and single Intel 5520 IOH, Intel Knights Ferry with D0 ED silicon (GDDR5 with 3.6 giga transfers per second [GTps], driver v1.6.501, flash image/micro OS v1.0.0.1140/1.0.0.1140-EXT-HPC, Intel Composer XE for MIC v048), and Nvidia C2050 (GDDR5 with 3.0 GTps, driver v270.41.19, CUDA 4.0).

References

1. M. Feldman, "TACC Steps Up to the MIC," *HPCwire*, 21 Apr. 2011; www.hpcwire.com/hpcwire/2011-04-21/tacc_steps_up_to_the_mic.html.
2. Nvidia, "Oak Ridge National Lab Turns to NVIDIA Tesla GPUs to Deploy World's Leading Supercomputer," *HPCwire*, 11 Oct. 2011; www.hpcwire.com/hpcwire/2011-10-11/oak_ridge_national_lab_turns_to_Nvidia_tesla_gpus_to_deploy_world_s_leading_supercomputer.html.
3. Intel, "Introducing Intel Many Integrated Core Architecture," press release, 2011; www.intel.com/technology/architecture-silicon/mic/index.htm.
4. Nvidia, "Next Generation CUDA Compute Architecture: Fermi," white paper, 2010; www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf.
5. L. Seiler et al., "Larrabee: A Many-Core x86 Architecture for Visual Computing," *ACM Trans. Graphics*, vol. 27, no. 3, 2008, pp. 18:1–18:15.
6. A. Heinecke and C. Trinitis, "Making TifaMMy Fit for Tomorrow: Towards Future Shared Memory Systems and Beyond," *Proc. 2011 Int'l Conf. High Performance Computing and Simulation*, IEEE Press, 2011, pp. 517–524.
7. H.-J. Bungartz and M. Griebel, "Sparse Grids," *Acta Numerica*, vol. 13, no. 5, 2004, pp. 147–269.
8. D. Pflüger, *Spatially Adaptive Sparse Grids for High-Dimensional Problems*, doctoral dissertation, Institut für Informatik, Technische Universität München, Germany, 2010.
9. A. Heinecke and D. Pflüger, "Multi- and Many-Core Data Mining with Adaptive Sparse Grids," *Proc. 8th ACM Int'l Conf. Computing Frontiers*, ACM, 2011, pp. 29:1–29:10.
10. A. Heinecke et al., "Extending a Highly Parallel Data Mining Algorithm to the Intel Many Integrated Core Architecture," *Proc. 4th Workshop UnConventional High Performance Computing*, Springer, to be published.

Alexander Heinecke is a research associate at the Technische Universität München (TUM), Germany; he previously interned for 10 months at Intel, where he worked with the Intel Knights Ferry prototype. His research interests include the use of multicore and many-core architectures in advanced scientific applications. Heinecke has an MSc in computer science from TUM. Contact him at heinecke@in.tum.de.

Michael Klemm is part of the Software and Services Group for the Developer Relations Division at Intel. His focus is on high-performance and throughput computing. Klemm has a Dr-Ing in computer science from the Friedrich-Alexander University, Erlangen-Nuremberg, Germany. His areas

of interest include parallel programming, performance analysis, and performance tuning. Contact him at michael.klemm@intel.com.

Hans-Joachim Bungartz is one of the directors of the Leibniz Supercomputing Centre; he is also a full professor of informatics and mathematics and holds the Scientific Computing chair at the Informatics Department of TUM. His research interests are in efficient algorithms in computational science and engineering, as well as high-performance computing. Bungartz has a Dr in computer science from TUM. Contact him at bungartz@in.tum.de.

cn *Selected articles and columns from IEEE Computer Society publications are also available for free at http://ComputingNow.computer.org.*