INVITED PAPER

Convergence of Recognition, Mining, and Synthesis Workloads and Its Implications

Future, more powerful, computers should interact with users to assess their needs, locate the data to meet those needs and process the data so it becomes understandable to the user.

By Yen-Kuang Chen, *Senior Member IEEE*, Jatin Chhugani, Pradeep Dubey, *Fellow IEEE*, Christopher J. Hughes, *Member IEEE*, Daehyun Kim, Sanjeev Kumar, *Member IEEE*, Victor W. Lee, Anthony D. Nguyen, *Member IEEE*, and Mikhail Smelyanskiy, *Member IEEE*

ABSTRACT | This paper examines the growing need for a general-purpose "analytics engine" that can enable next-generation processing platforms to effectively model events, objects, and concepts based on end-user input, and accessible datasets, along with an ability to iteratively refine the model in real-time. We find such processing needs at the heart of many emerging applications and services. This processing is further decomposed in terms of an integration of three fundamental compute capabilities—recognition, mining, and synthesis (RMS). The set of RMS workloads is examined next in terms of usage, mathematical models, numerical algorithms, and underlying data structures. Our analysis suggests a workload convergence that is analyzed next for its platform implications. In summary, a diverse set of emerging RMS applications from market segments like graphics, gaming, media-mining, unstructured information management, financial analytics, and interactive virtual communities presents a relatively focused, highly overlapping set of common platform challenges. A general-purpose processing platform designed to address these challenges has the potential for significantly enhancing users' experience and programmer productivity.

KEYWORDS | Algorithms; data structures; emerging applications; parallel architectures

Manuscript received March 16, 2007; revised October 19, 2007.
The authors are with the Microprocessor Technology Labs, Intel Corporation, Santa Clara, CA 95054 USA (e-mail: yen-kuang.chen@intel.com; jatin.chhugani@intel.com; pradeep.dubey@intel.com; christopher.j.hughes@intel.com; daehyun.kim@intel.com; sanjeev.kumar@intel.com; victor.w.lee@intel.com; anthony.d.nguyen@intel.com; mikhail.smelyanskiy@intel.com).

Digital Object Identifier: 10.1109/JPROC.2008.917729

I. INTRODUCTION

The performance of computers has improved dramatically in the past 40 years. Computers' ability to perform a huge number of computations per second has enabled many applications that have an important role in our daily lives, including a host of media-processing applications. Looking forward, we are all curious how future, even more powerful computers will change our lives.

It is extraordinarily difficult to accurately predict which next-generation applications will become popular (i.e., what will be the next "killer apps"). However, we believe these applications will also involve digital media and that another big leap in computing capability is needed to harvest and take full advantage of digital content. The ability to have computers intelligently understand and interpret data will help us in business, medicine, sociology, science, and personal hobbies.

There are at least three reasons why systems that can help us understand and interpret data will be important. First, the world's data outstrips our ability to comprehend it, much less take maximum advantage of it. According to the How Much Information project at the University of California at Berkeley,[1] print, film, magnetic, and optical storage media produced about 5 exabytes (EB) of new information in 2003. Further, the information explosion is accelerating. Market research firm IDC estimates that in 2006, 161 EB of digital content were created, and that will rise to 988 EB by 2010. To handle all this information, people will need systems that can help them understand

[1]http://www.sims.berkeley.edu/how-much-info-2003.

and interpret it. Search engines will not be enough. Tapping the Web alone, today's search engines often turn up thousands of documents in a single search, but many have minimal relevance. Moreover, 50 million new or changed Web pages are added every day. In addition to text, the world's data includes videos, photos, and various other kinds of media. We need computers to "see" data the way we do, identify what is useful to us, and assemble it for our review. The ability to have computers intelligently understand this data and help us use it in business, medicine, sociology, science, hobbies, and virtually every other realm of human endeavor could have enormous benefits and initiate countless revolutionary advances in human knowledge.

Second, the Web is shifting its focus from "data presentation to end-users" to "automatic data processing on behalf of end-users." Today's World Wide Web is a distributed data repository, a distributed compute infrastructure, and a composable service infrastructure. Most of the Web's current contents are intended for humans to read, not for computer programs to analyze. However, the Web has recently begun to transform into the "Semantic Web." According to Berners-Lee [2], the Semantic Web "adds logic to the Web." This will change the Web's functionality from "delivering data to users" to "processing data for users." Computers will use automated reasoning to help us understand and interpret structured collections of information via sets of inference rules.

Third, there is an inherent gap between a user's conceptual model of a problem they want solved and his/her computer's model of the problem. One of the primary goals of any computer system is to interact with its user and respond to user commands efficiently. As shown in Fig. 1, a major challenge to achieving this goal is bridging the gap between the user's and computer's model of the underlying problem to be solved. This gap is also known as Norman's gulf [23]. As noted by Norman in his seminal paper, a typical everyday problem requires multiple iterations of "execute and evaluate" between the user and the system. An iteration consists of the user's deciding which command he/she believes will result in the computer's solving the problem and constructing the command in a machine-readable format; then the computer's executing the command and generating user-readable output. Each such iteration normally narrows the modeling gap. Given the increasing complexity of emerging end-user compute scenarios ("find me this picture" rather than "add this number pair"), Norman's gulf has been growing. Fortunately, emerging computer usage models are introducing new ways to bridge Norman's gulf. Instead of involving the end-user in every iteration, an alternative is to depend on a computer's ability to refine model instances by itself. This allows a reduction in the number of interactions between a user and his/her computer, and therefore an increase in the system's efficiency.

The above three are examples of the tasks of an "analytics engine" that can model events, objects, and concepts based on what we show the computers and on the data accessible to them. Hence, we must be able to communicate with computers in more abstract terms (high-level concepts or semantics). We believe that an analytics engine must have the capability to construct, manipulate, and evaluate mathematical models. These capabilities can be classified as three distinct classes: recognition, mining, and synthesis (RMS). We call this classification the *RMS taxonomy* and describe it in detail in Section II. Briefly, recognition is the "what is." It is identifying that a set of data constitutes a model and then constructing that model. Mining refers to searching a data set, such as the Web, and asking "is it?" Synthesis is discovering "what if" cases of a model, i.e., the ability to create an instance of a model where one does not exist. Section II examines a set of workloads in terms of this taxonomy.

Emerging applications will likely iterate through different recognition, mining, and synthesis modules to build and use models. The iterations are necessary to refine models to bridge Norman's gulf. Such a refinement loop typically involves user *interaction*. Hence, we refer to these loops as iRMS loops.

Recognition, mining, and synthesis workloads often share some common kernels, mathematical models, numerical algorithms, and problem structures. We call this phenomenon workload convergence (Section III). As a result of this convergence, a diverse set of emerging RMS workloads presents a relatively focused set of common design challenges. We explore some of the key software and hardware challenges as well as possible avenues for overcoming these challenges (Section IV). A system designer who can overcome these challenges can construct a single architecture that can cost-effectively meet the needs of most or all RMS workloads. This is excellent news for everyone who will use, program, and design future computer systems because it will enable the building of faster systems with powerful software tools, both of which will enable brand new usages of computer systems.

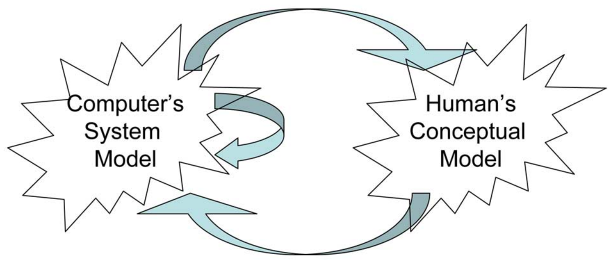


Fig. 1. ***Norman's gulf.***

II. INTERACTIVE RMS APPLICATIONS (iRMS)

A. RMS Taxonomy

Bridging Norman's gulf requires that applications build and use sophisticated models. The RMS taxonomy [8], [20] was proposed as a way to classify techniques used in such applications.

Recognition: Computers examine data and images and construct mathematical models based on what they "see." Depending on the data provided, that model could be of a valuable vase, a terrorist's behavior pattern, the right time to sell a particular type of stock, or the qualities needed by an actor to successfully play the part of Othello. This is a type of machine learning called recognition. Recognition is the "what is." It is identifying that a set of data constitutes a model and then constructing that model.

Mining: Once a computer has recognized the "what is" and turned that data into a model, the computer must be able to search for instances of the model. This is mining. Mining refers to searching a data set, such as the Web, and asking "is it?" to find instances of a model, such as good stock-trading opportunities or the best actors to play Othello.

Synthesis: Synthesis is discovering "what if" cases of a model. If an instance of the model does not exist, a computer should be able to create a potential instance of that model in an imaginary world. In other words, synthesis is the ability to create an instance of a model where one does not exist. For example, if a director is considering switching an actor in Othello, synthesis will show how that new actor would appear in the play and possibly predict the success of making the switch.

Beyond its use as a taxonomy, RMS offers a view of the underlying technologies. Traditionally we have treated "R," "M," and "S" components as independent application classes. For example, graphics applications used by animation movie studios to render high-quality animated movies are primarily synthesis applications. Similarly, data warehousing primarily involves mining. In contrast, emerging interactive applications use a combination of technologies that span the RMS spectrum. The remainder of this section examines this trend.

B. iRMS Loop

iRMS applications iterate through different recognition, mining, and synthesis modules to build, manipulate, and use models. Models are refined during each iteration of the loop, which we refer to as an iRMS loop. Fig. 2 shows the concept of iRMS loops.

The better computers are able to build models (recognition), the better computers should be at finding instances that fit these models and our needs (mining). For example, a robust model for a car will be able to locate its instances even when the car images are nonlinearly transformed or partly obscured.

Additionally, the better a model is constructed, the better that computers can manipulate the model in reality augmentation (synthesis). For example, the Digital Michelangelo Project demonstrated that computer-generated graphics are almost indistinguishable from photos when a model contains 4 million polygons with a resolution of 1 mm. Furthermore, those results of mining and synthesis can be fed back to the model construction process for further improvements in building the model (recognition). Finally, since today RMS technologies are usually not perfect, user interaction is needed to guide the refinement process.

C. iRMS Nested Loops

A user interacts with a computer via an iRMS "visual computing" loop, where a representation or instance of the model is presented in a human-accessible format (e.g., images or video). To address the growing Norman's gulf, the user needs to be taken out of the model refinement process as much as possible. For each iteration of the visual computing loop, one can add one or more iterations of an iRMS analytics loop, which does not involve end-user visualization and instead depends on a machine's ability to refine model instances by itself. Fig. 3 shows a nested iRMS loop. As the inner loop of user-independent machine analytics becomes more efficient at bridging the model gap for increasingly complex models, the number of visual outer loop iterations will drop and will focus on higher levels of abstraction. It is important to note that the latency of each visual outer loop iteration is fundamentally limited by human response-time limitations (speed of keyboard

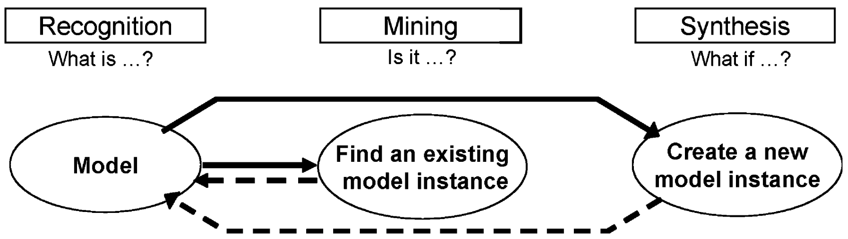


Fig. 2. *iRMS loop.*

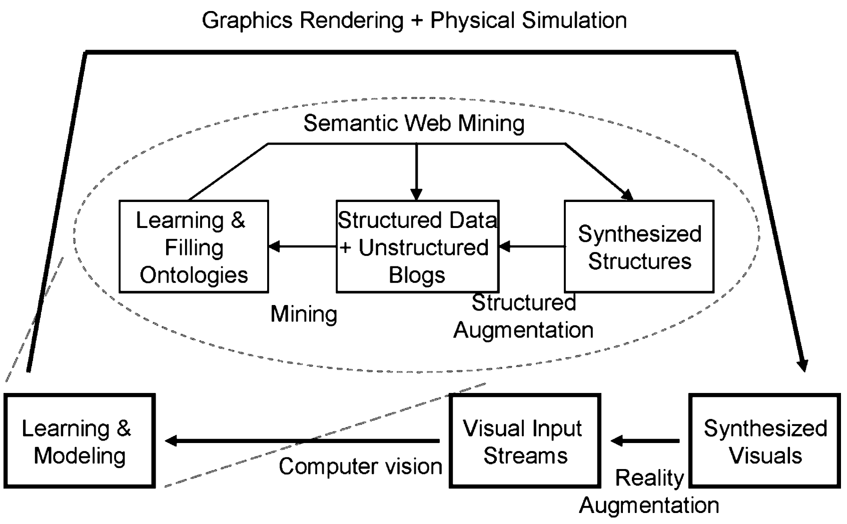


Fig. 3. ***iRMS nested loop.***

input, etc; recall, a typical user has very limited cognitive abilities). Also, the quality requirements of the outer loop are limited by typical human sensory limitations such as "real-enough photos." On the other hand, the inner loop latency is purely limited by the data-processing abilities of the compute infrastructure. Therefore, in principle, the inner loop can iterate much faster than a typical outer loop with a human in the loop. For example, millisecond trades are possible, as no human keystrokes are involved.

The emergence of the Semantic Web is also very relevant in this context. As noted by Berners-Lee [3], the primary goal of the Semantic Web is to add logic to the Web. This implies that processing requirements growth will increasingly shift from "visual computing" to "analytics." That is, there will be multiple inner loop iterations of nonvisual computing (i.e., analytics) for every outer loop iteration of visual computing. This potential shift further assumes that the underlying software stack (XML/RDF/OWL-like) proposed by Berners-Lee [2] and reproduced in Fig. 4 gains traction in the developer community and that applications such as virtual environments (e.g., Second Life and Sony's Home) gain acceptance among end-users. Fig. 5 illustrates a virtual environment as an instance of a nested iRMS loop as described above.

D. Examples of iRMS Applications

Below are other examples of applications that use a combination of RMS techniques and that include iRMS loops and nested loops.

Medicine: Fig. 6 shows how, through RMS, a tumor could be:

1) recognized as a model;
2) identified through mining patient data as the type of tumor in a particular patient;
3) synthesized in a way that would predict the effects of the tumor's progression for a particular patient and whether treatment is advisable.

Synthesis could also be used as a means to determine the efficacy of various treatment options. One could take what is known about each treatment model and perform synthesis with the patient's medical history and condition. For instance, on a cancer tumor, synthesis could be used to simulate the effects and prognosis of treatments by chemotherapy, radiation, surgery, combined approaches, or no intervention at all. By examining the outcomes of each treatment, both the doctor and the patient could decide the best course of action.

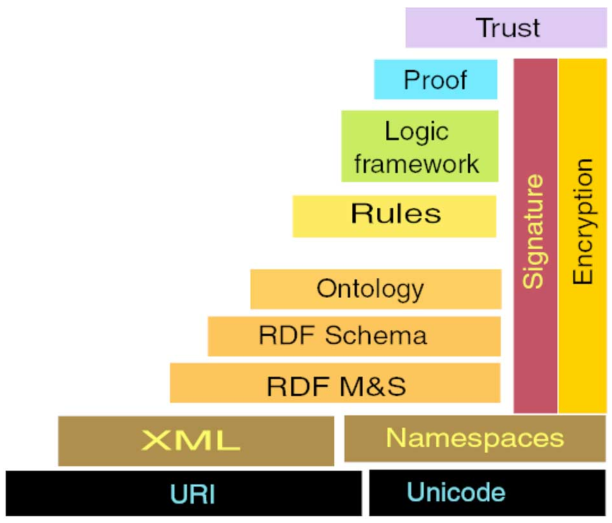


Fig. 4. ***Semantic Web software stack.***

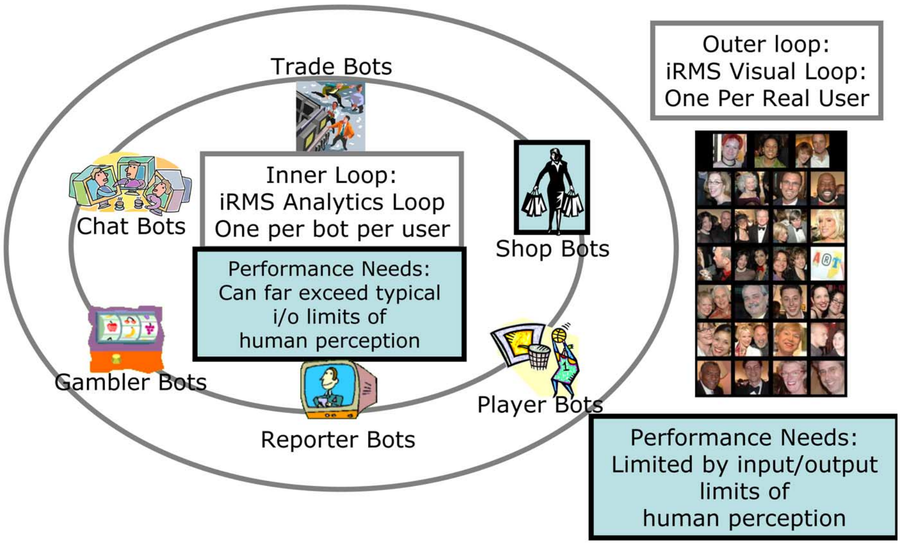


Fig. 5. ***A virtual environment (e.g., Second Life) as an iRMS nested loop instance.***

In the near future, individual genetic profiles may enable "personalized medicine." Using RMS, a patient's genetic profile could be modeled and kept on file. When a patient comes in with a specific condition, such as high blood pressure, a doctor could mine drug databases using the patient's profile for the best drug options. The doctor could then test through synthesis for possible reactions by that particular patient to the various drugs suggested. With technology like this, doctors could more safely prescribe medicines.

Investment: Selecting smart equity investments involves more than analysis of a company's financials. An investor should consider industry trends and a wide spectrum of potential factors that could include currency rates, trends in buying habits, oil prices, recent research on Asian attitudes toward American foods, and much more. There is so much possible data to consider that in the end, people often just rely on standard indicators, such as a stock's price-to-earnings ratio, recommendations by friends or a broker, and their gut feeling.

RMS could radically change that (see Fig. 7). Using a model of a successful investment, people could mine a dataset for other potential successful investments. Synthesis could then enable a person to examine "what if's" having to do with various investment periods and the

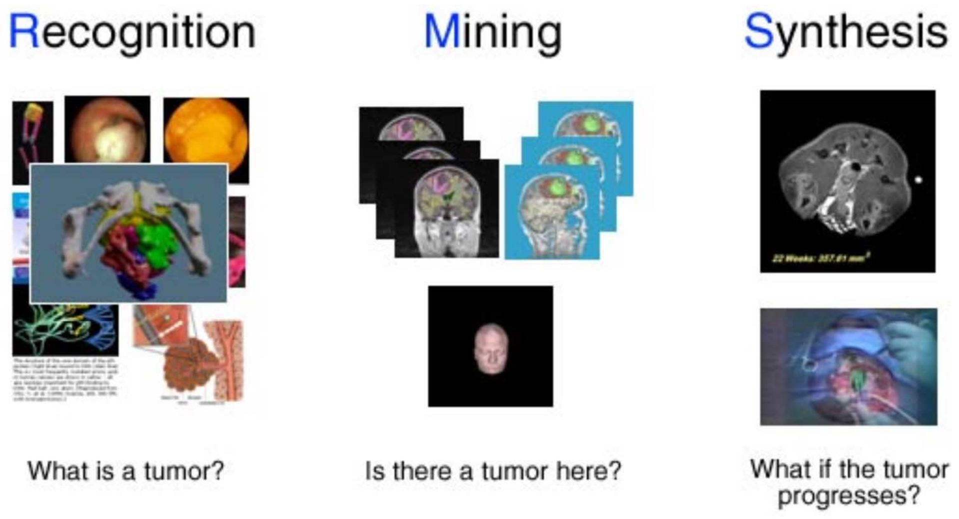


Fig. 6. ***An example of how RMS might be used in medicine.***

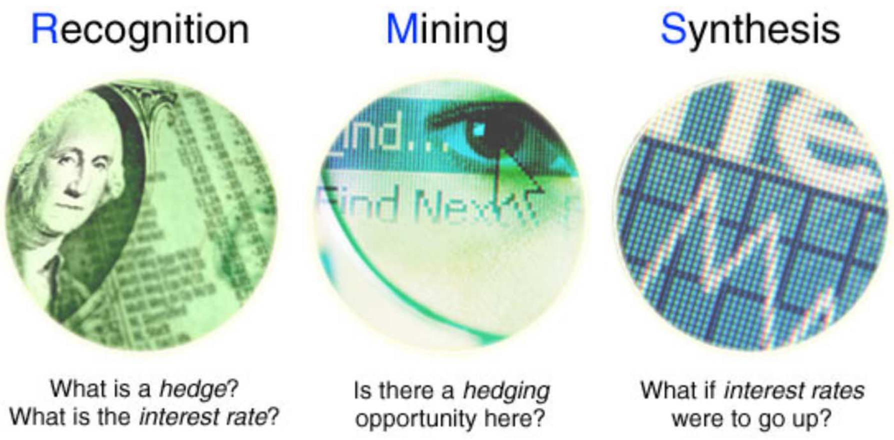

Recognition
Mining
Synthesis
What is a *hedge*?
What is the *interest rate*?
Is there a *hedging* opportunity here?
What if *interest rates* were to go up?

Fig. 7. ***RMS could potentially be used to "mine" for instances where, according to a particular hedge fund's strategy, it would make sense to invest. Synthesis could be used to predict what might happen to a particular hedge fund investment if certain conditions, such as prevailing interest rates, changed.***

potential effects of various events, such as a rise in interest rates or one company acquiring another. While RMS would not take all the uncertainty out of an investment, it could allow investors to make use of a much larger data set in determining which equities to buy. RMS could create new expectations for what better portfolio-management software should do.

Business: Successful businesses with large cash reserves often begin looking for acquisitions to broaden their product lines and their markets. Some work out; others do not. Some take years to pay off; others end up in divestiture. Even though financial analysts and other specialists pore over the prospective company's books for months before a transaction is completed, the result can still go sour over intangibles and unexpected turns of events. Through RMS, many of these intangibles could become tangible and unexpected turns of events could be anticipated. Using an acquisition model and mining the vast amount of data on the two companies and past acquisitions, outcomes could be more accurately predicted using synthesis. Other acquisition targets could also be more accurately "tried" before announcing any interest or a potential purchase.

Small businesses could profit from RMS as well. Independent retailers, such as a local auto parts store, have limited shelf space and need to optimize their inventory for both better profitability and customer service. Using a model of what a good inventory item is (good margin, fast mover), a local parts store could mine national car service databases by manufacturer to determine what parts to carry.

Surveillance could be another key business use of RMS. Today's surveillance cameras simply record what they see. Imagine if their images were run through a computer that could recognize and provide the right kind of alert for each instance of trouble. For example, a bank's surveillance system, trained through models to recognize a gun, could immediately alert police when someone has entered the bank with one. On a larger scale, RMS could help monitor freight. Today, only 4% of the container traffic coming into the United States on ships is inspected. Imagine if through radio-frequency ID tags, surveillance cameras, biological sensors, and other monitoring systems, computers could monitor imports using a database of models to identify which might require an alert and further inspection.

Another application for business could be hiring. Through recognition, a model of a successful employee for a particular job position could be created. Using mining, all particular potential candidates who fit this model could then be identified. Finally, through synthesis, an idea of how each candidate might perform in a number of job-specific situations could be examined. Hours of sourcing and interviewing could be reduced to interviewing only a few candidates who, according to the synthesis results, "performed" best in the simulated job situations.

Home: As the popularity of digital audio, photography, and video grows, so does the number of MP3 files, photos, and clips on people's hard drives. Having busy lives, most people do a poor job of naming and categorizing these files. The result is a great deal of time and frustration searching folders for particular audio files, photos, or video clips.

In the future, it is only going to get worse, especially as hard drives continue to grow in capacity and people continue to save digital media onto them in a haphazard manner. Imagine if, using RMS, a person could easily assemble for a twenty-fifth wedding anniversary a collection of photos and video clips of just the married couple. Through recognition and data mining, these files could be easily and quickly found among hundreds of thousands of files. Using synthesis, someone could even surprise the couple by showing what they might look like on their fiftieth anniversary or how they would look now in their original wedding photos.

Consider, too, how RMS might be used by people as they gain access through the Web to entire libraries of music, film, and television shows. It could unleash new channels of creativity and investigation as people use synthesis to see what one film star might have been like in another film star's part, or how using a different pitcher might have changed the outcome of a baseball game. Or imagine shopping on the Web and being able to use synthesis to interactively combine models of you and models of various articles of clothing. You'll be able to very quickly "try on" lots of outfits and colors before you ever click "buy"—and accurately determine how they fit.

Another excellent use of RMS would be information monitoring and sorting to help people stay current with a particular subject or interest. For example, a person's computer could constantly model specific information to look for based on what a person views and uses on their computer. This would be somewhat similar to how Amazon and other vendors on the Web today make product recommendations based on buying habits, but RMS would enable it to be applied to a much bigger universe of information.

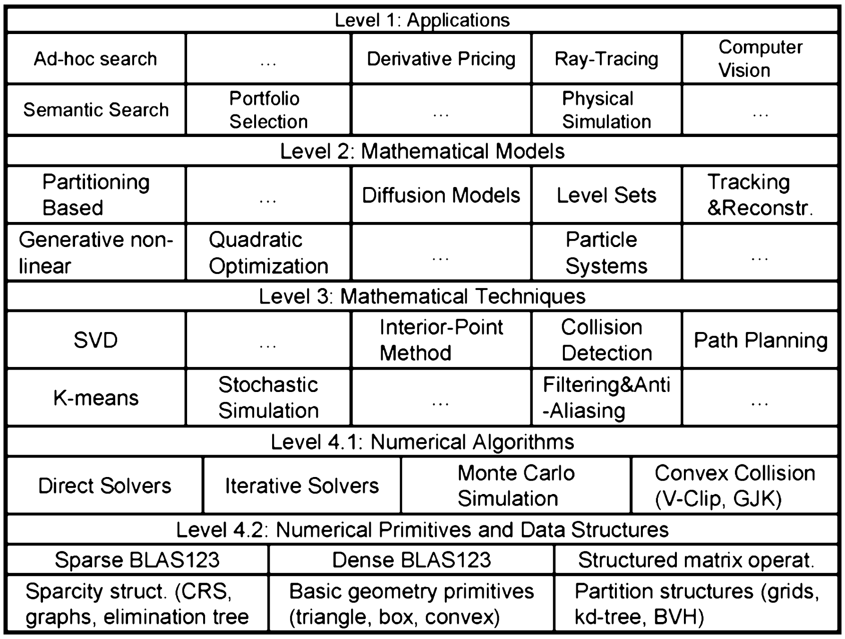
This universe could even include the multitudes of blogs on the Internet today. Imagine having your computer monitor and collect on your hard drive all the items (text, video, photographs) in this information universe that should be of interest to you. Using models, a computer could mine the entire Web continuously for information of value to a person's business, educational pursuits, and hobbies.

In short, RMS can be used independently in applications, as graphics has been historically separate from mining. However, an essential aspect of RMS is that the components R, M, S are not to be viewed in isolation. An application is more powerful when it can integrate multiple components of RMS, especially if it has the ability to loop back and forth between the components.

III. WORKLOAD CONVERGENCE

The previous section introduces the growing need for a general-purpose analytics engine and how the analytics tasks can be classified as being R, M, and/or S. We therefore refer to workloads run by such an engine as RMS workloads. These workloads come from a broad spectrum of domains, such as visual computing and medicine. This section shows that RMS workloads from any domain can be decomposed to a set of mathematical models and techniques, and those can be further decomposed to numerical algorithms and numerical primitives on data structures. We use some concrete examples to show that there is significant overlap between workloads from different domains even at the level of mathematical models, and almost complete overlap at the numerical algorithms and the primitive's levels. We call this phenomenon workload convergence.

Fig. 8 presents an RMS workload component hierarchy consisting of four levels: 1) RMS applications, 2) mathe-



Level 1: Applications				
Ad-hoc search	…	Derivative Pricing	Ray-Tracing	Computer Vision
Semantic Search	Portfolio Selection	…	Physical Simulation	…
Level 2: Mathematical Models				
Partitioning Based	…	Diffusion Models	Level Sets	Tracking &Reconstr.
Generative non-linear	Quadratic Optimization	…	Particle Systems	…
Level 3: Mathematical Techniques				
SVD	…	Interior-Point Method	Collision Detection	Path Planning
K-means	Stochastic Simulation	…	Filtering&Anti-Aliasing	…

Level 4.1: Numerical Algorithms			
Direct Solvers	Iterative Solvers	Monte Carlo Simulation	Convex Collision (V-Clip, GJK)

Level 4.2: Numerical Primitives and Data Structures		
Sparse BLAS123	Dense BLAS123	Structured matrix operat.
Sparcity struct. (CRS, graphs, elimination tree	Basic geometry primitives (triangle, box, convex)	Partition structures (grids, kd-tree, BVH)

Fig. 8. *RMS workload component hierarchy.*

matical models that use mathematical language to describe the behavior of each application, 3) mathematical techniques, which are the set of mathematical tools used to solve the model, 4.1) numerical algorithms, and 4.2) primitive operations and data structures that are used to support robust implementation of the mathematical models and techniques. In the following discussion, we map applications from three example RMS workload domains, to our workload component hierarchy: Web mining, financial analytics, and visual computing.

A. Example RMS Workload Domains

Fig. 9 is an expanded version of Fig. 8 that focuses on the first three levels of the hierarchy for each of our example workload domains.

Web Mining: The Web mining workload domain shown in Fig. 9(a) uses data mining to analyze and discover interesting statistical patterns of a user's usage of data on the Web, with the goal of categorizing and classifying new data [6]. Web mining applications spend more effort on the mining part of the iRMS loop. One example of a key Web mining application is collaborative filtering, a basis of modern recommender systems. Collaborative filtering automates predictions of the interests of a user by collecting and filtering taste information from many users. Another example of a Web mining application is named *entity recognition information extraction*. This application locates and extracts atomic text elements in documents and classifies them into predefined categories such as the names of persons, organizations, and locations.

These and other data-mining applications rely on a broad set of mathematical models that classify data. In classification, one starts off with a set of training observations that map instance vectors d_k to known categories σ_k. The problem is to determine the category x_i to which a given new observation vector should belong. For example, to solve a discriminative model of a classifier, one can use a support vector machine (SVM), which

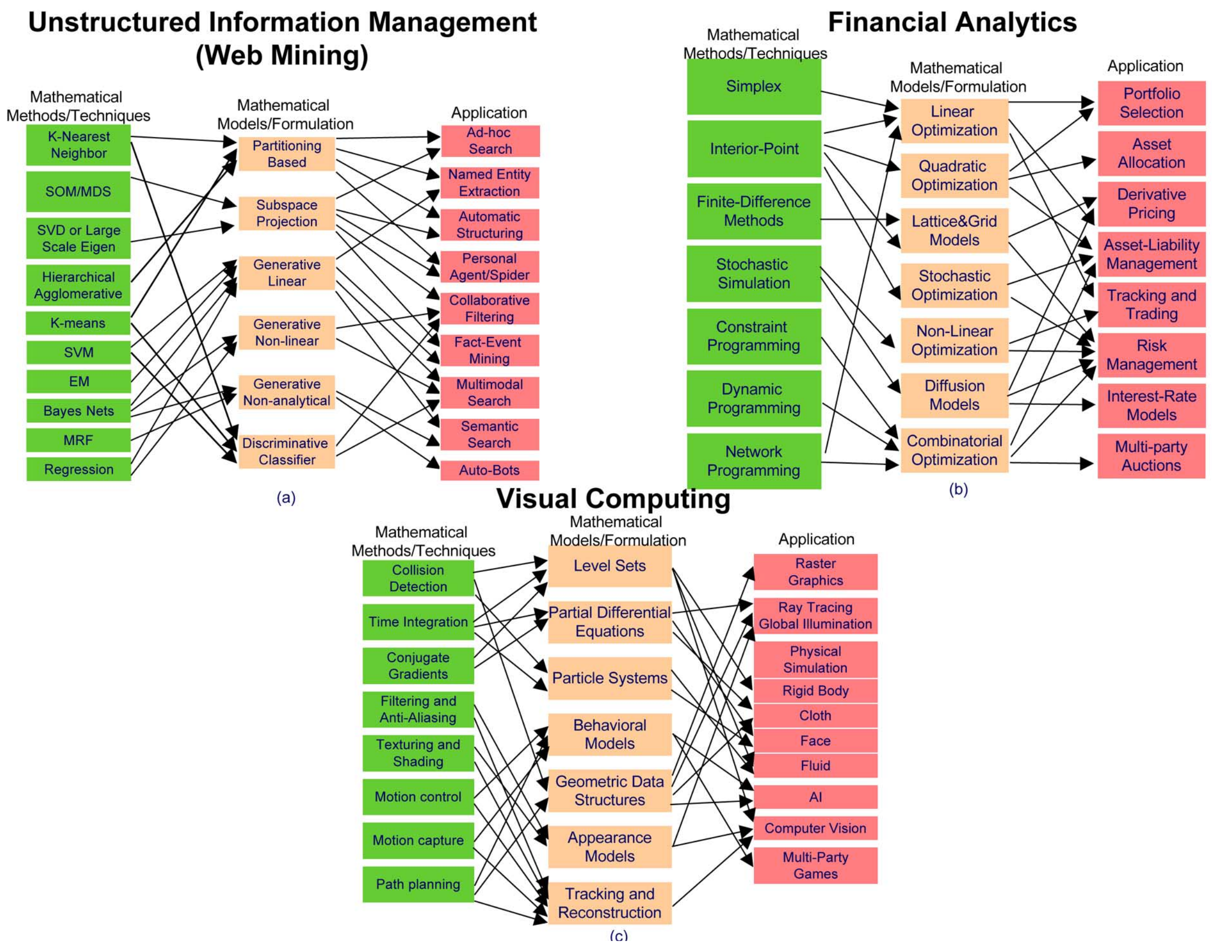


Fig. 9. ***Three example RMS workload domains, including a set of important applications in each domain, and their underlying mathematical models and techniques.***

reduces a classification model to a quadratic optimization problem of the form

$$\begin{aligned} &\text{Maximize}_x \quad -\frac{1}{2}\sum_{i,j}\rho_{i,j}\sigma_i\sigma_j x_i x_j + \sum_i x_i \\ &\textit{subject to} \quad \sum_i \sigma_i x_i = 1, \quad 0 \le x_i \le C, \ \forall i = 1,\ldots,n. \end{aligned}$$

The solution to this optimization problem will separate the observation vectors into categories with the maximum separating distance.

Financial Analytics: The financial analytics workload domain shown in Fig. 9(b) uses mathematical finance, numerical methods, and computer simulations to make trading, hedging, and investment decisions as well as to facilitate risk management of those decisions [7]. As discussed earlier, financial analytics applications map to all three parts of the iRMS loop.

Derivative pricing is an important example financial analytics application. The goal of derivative pricing is to find the fair price of a financial derivative of one or more underlying assets [18]. The modern mathematical formulation of derivative pricing uses a Black–Scholes diffusion model [5], [21], which models a derivative as a stochastic partial differential equation. Solving a Black–Scholes diffusion model relies on a wide range of mathematical techniques, such as lattice methods and stochastic simulation.

Given n derivatives, each with expected return μ_i and variance σ_i, one seeks to select an optimal portfolio with maximum expected return while keeping risk under a threshold, R. Such a portfolio is modeled using a Markovitz portfolio optimization problem, which reduces portfolio selection to a quadratic optimization problem of the form

$$\begin{aligned} &\text{Minimize}_x \quad \sum_{i,j}\rho_{ij}\sigma_i\sigma_j x_i x_j \\ &\textit{subject to} \quad \sum_i x_i = 1, \ \sum_i \mu_i x_i \ge R, \ x_i \ge 0, \ \forall i = 1,\ldots,n. \end{aligned}$$

Here $\rho_{i,j}$ is the correlation between two derivatives, which characterizes the degree of independence between these derivatives. For example, a correlation of one implies a linear relationship between two derivatives, while a correlation of zero implies independent derivatives. Quadratic optimization problems that we encountered in Web mining and finance are solved using the interior-point method [24].

Visual Computing: The main task of applications in the visual computing workload domain shown in Fig. 9(c) is to acquire, analyze, and synthesize visual data [1], [25]. They often spend most of their time in the synthesis part of the iRMS loop.

One example visual computing application is rendering using global illumination. This can be used in computer games or in generating movie special effects. It recognizes where a player/actor is and where the light sources are, and simulates in real-time all the points or rays of light. Another example is physical simulation, which models systems of objects such as rigid bodies, cloth, and fluids and their motion according to Newton's laws of dynamics.

Performing physically correct simulation of the interaction of light and physical objects relies on a rich set of mathematical models. For example, appearance models are used to model and simulate objects made up of different materials. Also, particle system models are used to simulate fuzzy physical objects such as fire, explosions, smoke, flowing water, sparks, falling leaves, clouds, fog, and snow. Mathematical models used in visual computing may or may not be physically accurate. While physical accuracy is important in film-production-quality simulation, games can typically tolerate rougher approximations.

A wide variety of mathematical techniques is used to solve models in visual computing. One example is collision detection, which is used to determine collisions between two geometries, such as a ray and a triangular mesh in the case of rendering, or two convex geometries in the case of physical simulation.

B. Numerical Algorithms

We have shown the decomposition of a variety of RMS applications into mathematical models and techniques. Robust numerical algorithms are required to solve these mathematical models. Therefore, in Fig. 10, we decompose those models and techniques further, into numerical algorithms, and even further into numerical primitives on data structures. Virtually all RMS applications map to (i.e., spend much/most of their time in) a very small, common set of numerical algorithms, and therefore to a small, common set of numerical primitives and data structures. We call this phenomenon *RMS workload convergence*. While there is some overlap between RMS applications from different workload domains, even at the levels of mathematical models and techniques, the small footprint of the final two levels of the workload component hierarchy highlights RMS workload convergence.

Fig. 11 provides an illustrative example of workload convergence using several applications from our example RMS workload domains. Arrows in this figure connect an example application with a subset of numerical algorithms to which this application maps.

We observe that most numerical algorithms used in RMS applications can be divided into the following four categories: direct solvers, iterative solvers, Monte Carlo simulation, and convex collision-detection methods. We now briefly describe each of these categories. In the following section, we

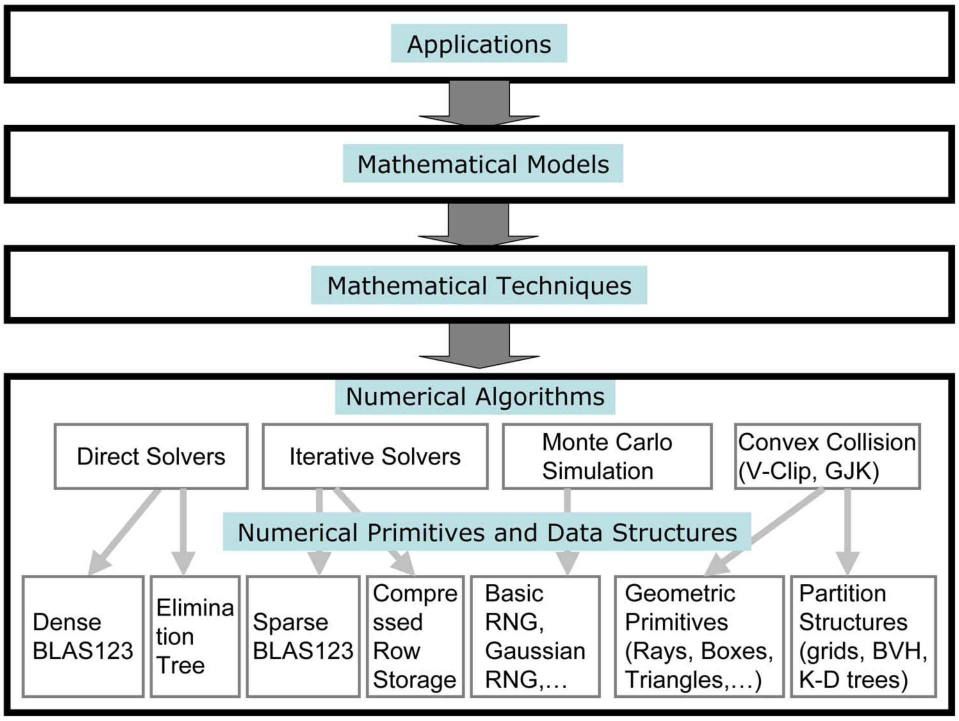


Fig. 10. ***Decomposition of mathematical models and techniques into numerical algorithms and primitives.***

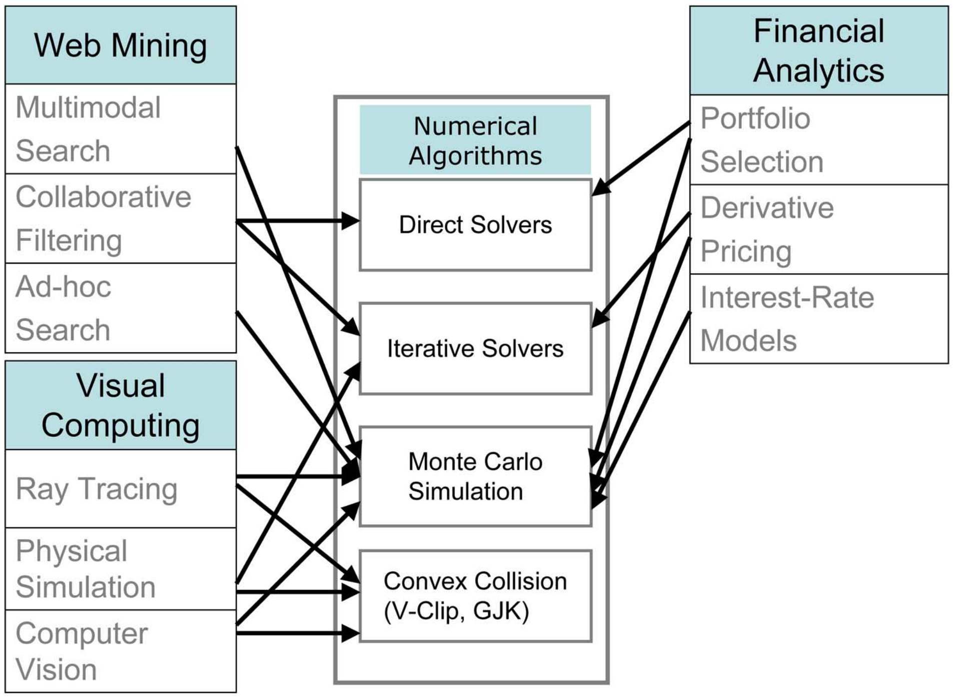


Fig. 11. ***Workload convergence. This example illustrated several applications from three RMS workload domains, which map to a small, common set of numerical algorithms.***

will discuss the lower level numerical primitives and data structures.

Direct and Iterative Solvers: Numerical solutions to optimization problems and partial differential equations rely on solving linear systems of equations $Mx = b$. The two primary methods of solving linear systems are direct and iterative [15], [26].

Direct solvers for linear systems of equations involve the following two steps:

1) LU factorization, which decomposes matrix M into lower triangular submatrix L and upper triangular submatrix U, such that $M = LU$;
2) triangular solver, which uses the results of factorization to solve a system of linear equations $LUx = b$, using the following two substeps:
    a. forward solver, which solves $Ly = b$;
    b. backward solver, which solves $Ux = y$.

Direct methods use low-level dense linear algebra operations, such as dense matrix–matrix product and dense matrix–vector product. Efficient implementations of these low-level primitives are crucial for achieving high performance on direct solvers.

Two main drawbacks of direct solvers are their high computational complexity and high storage requirements. This makes them prohibitively expensive for very large problems. For such problems, iterative methods are preferred. There are many iterative methods; however, the RMS applications we have studied to date rely only on a small subset of these methods, in particular successive overrelaxation (SOR) and conjugate gradient. Examples of SOR use are solving linear complementarity problems that arise in rigid body simulation and in pricing American-style derivatives. An example of conjugate gradient use is solving partial differential equations that arise in physical simulation of cloth, fluids, and face.

Fig. 12 shows an example of the conjugate gradient method. It starts with an initial guess to the solution x_{initial}. The core of the method is the main optimization loop, which updates the vector x at each iteration until convergence to the optimal solution is achieved.

While direct methods rely on dense matrix operations, the core of all iterative methods is sparse matrix–vector product and sparse backward solve.

```
r = $Mx_{initial} - b$
for $i=0$ .. num_iterations
  solve preconditioner: $z = M^{-1}r$
  inner product: $v_i = <r,z>$
  update: $p = z + p * (v_i/v_{i-1})$
  matrix-vector product: $q = Ap$
  inner product: $\pi = <p, q>$
  update: $x = x - v*\pi$
          $r = r - q*\pi$
end
```

Fig. 12. ***Example of a conjugate gradient iterative solver.***

Monte Carlo Method: A key numerical technique for solving problems in combinatorial and stochastic optimization is Monte Carlo simulation. Monte Carlo simulation iteratively evaluates complex mathematical models using sets of random numbers as inputs. This is especially useful where the complexity of the mathematical model precludes representing the function to be solved analytically.

The idea of Monte Carlo can be demonstrated using evaluation of a simple integral $I = \int_a^b f(x)dx$. Using the definition of the integral, it can be approximated as $I = \int_a^b f(x)dx \approx ((b-a)/N)\sum_{i=1}^{N} f(x_i)$. Instead of directly computing the integral, Monte Carlo simulation takes many random samples x_i, computes $f(x_i)$ for each sample, and averages these contributions to approximate the integral.

Monte Carlo simulation is widely used in computational finance for pricing complex option derivatives where analytical methods are not applicable [14]. An option is a financial contract between the buyer and the seller, where the buyer of the option has the right, but not the obligation, to buy an agreed quantity of a particular stock S from the seller of the option at the expiration date T for a certain price. Given the nondeterministic nature of an underlying stock, one uses Monte Carlo simulation to find the price of an option as follows.

```
for $i = 0 \ldots N$
  1. Sample random stock $S$ price from time $t = 0$ to $t = T$
  2. Calculate the payoff from the derivative, $p_i$
end
3. Calculate expected payoff: $P = \sum_{t=0}^{N} p_i$
4. Discount $P$ at risk-free rate to get option price.
```

Efficient implementations of Monte Carlo simulation rely on a high quality random number generator.

Collision Detection: Collision detection is an important numerical technique that is at the core of many visual computing applications. Collision detection is used to determine whether two objects, represented as geometries, are in contact or interpenetrate each other, such as a ray and a triangular mesh in ray-tracing, or a cylinder and an axis-aligned box in physical simulation. Collision detection is used to calculate collision points, collision trajectories, and collision times [10].

There are two types of geometries: convex and nonconvex (see Fig. 13). An object is convex if, for every pair of points within the object, every point on the straight line segment that joins them is also within the object. For example, a triangle is convex, a solid cube is convex, but anything that is hollow or has a dent in it is not convex.

While there exists a number of specialized and optimized routines for determining the collision between a ray and a triangle, collision detection among more complex convex geometries is commonly performed using the

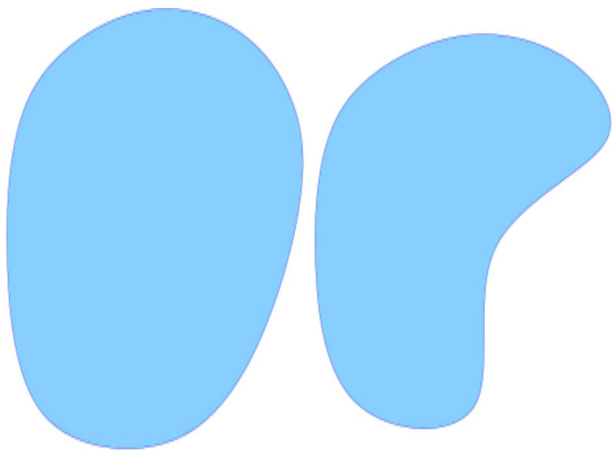


Fig. 13. ***Convex and nonconvex geometries.***

GJK algorithm [13]. GJK iteratively computes the distance between two convex geometries until the minimum distance is found. In the case of nonconvex geometries, the frequently used methods are 1) to break the nonconvex geometry into several convex geometries and use convex collision methods on each and 2) to represent the geometry using a triangulated surface and then use a general algorithm for colliding triangular meshes, which again involves colliding convex geometries.

Collision detection algorithms rely on acceleration data structures to significantly reduce the number of collision tests, as we describe in the next section.

C. Numerical Primitives and Data Structures

The small set of numerical algorithms common to RMS applications is built on an equally small set of numerical primitives operating on data structures, the last level of the RMS workload component hierarchy (see Figs. 8 and 10). In this section, we provide several illustrating examples of these primitives and data structures.

BLAS: The basic linear algebra subprograms (BLAS) are standard building blocks for performing basic vector and matrix operations. The Level 1 BLAS performs scalar–vector and vector–vector operations, the Level 2 BLAS performs matrix–vector operations, and the Level 3 BLAS performs matrix–matrix operations. Both direct and iterative solvers heavily use BLAS routines.

Matrix Storage Formats: Many underlying problems in RMS applications can be represented by large matrices with very few nonzero elements, referred to as sparse matrices. To avoid storing a large number of zeros, sparse matrices are often compressed to yield significant savings in memory usage. Several sparse matrix formats exist [9], [26], such as compressed row storage (CRS), jagged diagonal format, and compressed diagonal storage format. Each format takes advantage of a specific property of the sparse matrix and therefore achieves a different degree of space efficiency. The CRS format is perhaps the most widely used format in RMS applications (Fig. 14).

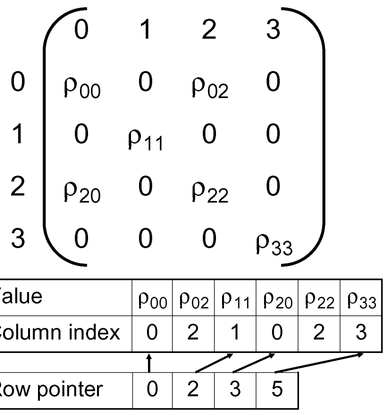
As an example, consider the portfolio optimization problem discussed earlier. If every derivative is correlated with every other derivative, a dense correlation matrix is used. In a more realistic scenario, each derivative is only correlated with a few others, so a sparse correlation matrix is used.

CRS uses three vectors to represent a matrix. The value vector holds the nonzero values, the column index vector indicates the column index for each element in the value vector, and the row vector indicates each row's first element in the value vector.

All primitive sparse matrix operations are performed using their CRS format directly.

Supernode and Elimination Tree: We further consider two other data structures that are paramount to efficient implementations of direct linear solvers: supernodes and elimination trees [22]. A supernode is a set of contiguous columns in a matrix whose nonzero structure consists of a dense triangular block on the diagonal and an identical set of nonzeroes for each column below the diagonal. An example of a matrix partitioned into supernodes is given in Fig. 15. In this example, a 13 × 13 sparse matrix (where zero entries are empty and nonzero entries are indicated with an "x") is broken down and stored in six supernodes (sn1, sn2, . . ., sn6). Columns 1 and 2 with the same nonzero structure form supernode 1 (sn1). Similarly, columns 5 and 6 form supernode 3 (sn3). Since all columns in a supernode have an identical nonzero structure, in practice, nonzero elements of supernodes are compactly compressed into and stored as a dense matrix.

An elimination tree is a task dependence graph that characterizes the computation and data flow among the supernodes of a sparse matrix during Cholesky factorization and triangular solving. It is defined as follows:



$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & \rho_{00} & 0 & \rho_{02} & 0 \\ 1 & 0 & \rho_{11} & 0 & 0 \\ 2 & \rho_{20} & 0 & \rho_{22} & 0 \\ 3 & 0 & 0 & 0 & \rho_{33} \end{array}$$

Value	ρ_{00}	ρ_{02}	ρ_{11}	ρ_{20}	ρ_{22}	ρ_{33}
Column index	0	2	1	0	2	3

Row pointer	0	2	3	5

Fig. 14. ***Matrix in CRS format.***

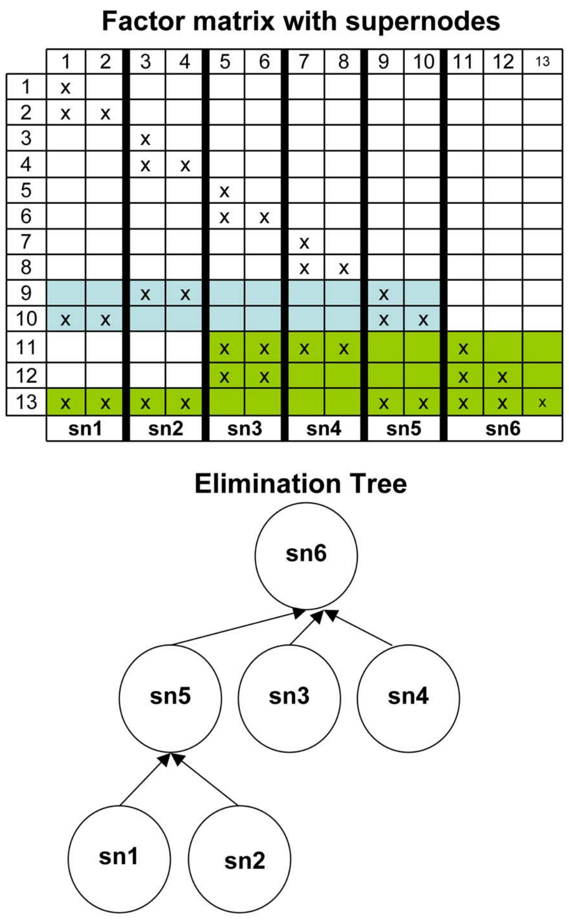


Fig. 15. ***Example of supernodal matrix structure and corresponding elimination tree.***

$\text{parent}(sn_j) = min\{sn_i | i > j$ and at least one of the elements of sn_j that correspond to the diagonal block of sni is nonzero$\}$. In other words, the parent of supernode j is determined by the first subdiagonal nonzero in supernode i.

Fig. 15 also shows an example of the elimination tree for the matrix. We see that there is an edge between sn1 and sn5 because, as the shaded portion of the figure shows, the second row of the 2 by 2 diagonal block of sn5 corresponds to the nonzero row 10 in sn1. Similarly, there is an edge between sn3 and sn6 because the first two rows of sn6 correspond to nonzero elements in rows 11 and 12 of sn3.

Spatial Partitioning Data Structures: Efficient implementation of key visual computing applications hinges upon several key acceleration data structures. These data structures partition the space of the underlying scene-graph to significantly reduce the number of collision tests needed.

For example, in ray-tracing, an acceleration data structure allows us to efficiently determine the path a ray takes through space and to test it only against those triangles that are in the vicinity of the ray's trajectory [25]. Similarly, in collision detection, an acceleration data structure spatially separates simulated objects such that expensive collision tests only occur between objects that are in each other's vicinity. Fig. 16 illustrates a few of the most frequently used spatial partitioning data structures: uniform grids, quadtrees [octrees in three dimensions (3-D)], and kd-trees.

These data structures partition space into a number of cells, and objects are mapped into the corresponding cells. While uniform grids partition space into equal-sized cells, octrees and kd-trees partition space into uneven-sized cells. This allows octrees and kd-trees to provide a tighter fit for the objects using the same number of cells as a uniform grid, or the same fit with fewer cells. Although kd-trees are harder to implement and are more expensive to build than octrees, they provide a tighter fit than octrees: while octrees recursively subdivide each cell into eight equal-size subcells, kd-trees subdivide each cell using axis-aligned splitting planes into arbitrary size subcells.

In summary, this section describes several numerical primitives and data structures that are commonly used across RMS applications to enable high-performance implementations. It is good news that we can identify a critical set of primitives and data structures—it means that a performance library designer can optimize these operations and data structures and incorporate them into tuned libraries, which will then accelerate many RMS applications.

IV. IMPLICATIONS FOR SYSTEM DESIGN

As just described, convergence of the mathematical models, numerical algorithms and primitives, and data structures in many RMS applications constitutes what we call workload convergence. Workload convergence has implications for programmers, users, and system designers. From a system designer's point of view, workload convergence is excellent news—it means that a large set of important workloads presents a relatively small set of common challenges. For example, we may cost-effectively

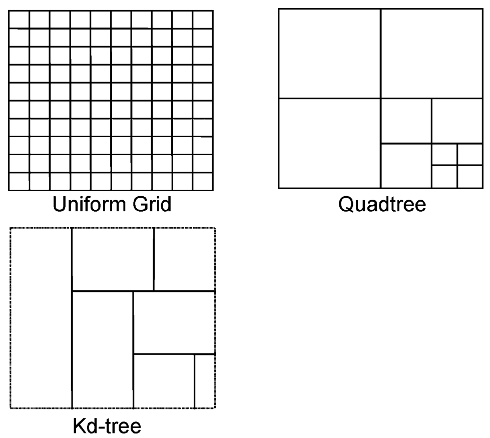


Fig. 16. ***Spatial partitioning data structures.***

design a single architecture capable of meeting the needs of most or all RMS workloads. We refer to this as system design convergence. In this section, we discuss the computation requirements of RMS workloads and explore how to provide the necessary compute power.

Computation Requirements: Most RMS workloads have very high compute requirements, and thus can benefit from orders of magnitude acceleration. Fig. 17 shows the computation requirements of some RMS applications for a variety of usage scenarios (i.e., different image/video resolutions, frame rates, grid resolutions for fluids, and number and type of assets). All of these applications have important usage scenarios with computation requirements approaching or even exceeding a teraflop, and some have requirements far beyond a teraflop. When we consider that future workloads will likely combine multiple of these applications, the need for large acceleration becomes even greater. Acceleration of multiple orders of magnitude will enable interactivity or even allow some applications to become real-time. The recent introduction of multicore processors, and the promise of many-core processors to come, means that such acceleration may be possible.

To harness the compute power of multicore and many-core systems, applications typically are parallelized via software threading. Fortunately, all RMS workloads that we have examined to date contain copious amounts of thread-level parallelism (although it is not always coarse-grained or easily extracted). That is, they have enough thread-level parallelism to see performance benefits from having at least tens of threads running concurrently. For example, Fig. 18 shows the parallel scalability of some RMS applications on a simulated 64-core chip multiprocessor. Therefore, while the compute requirements of most RMS workloads are well beyond what today's systems can provide, workload convergence means that the systems of tomorrow have the potential to meet those requirements.

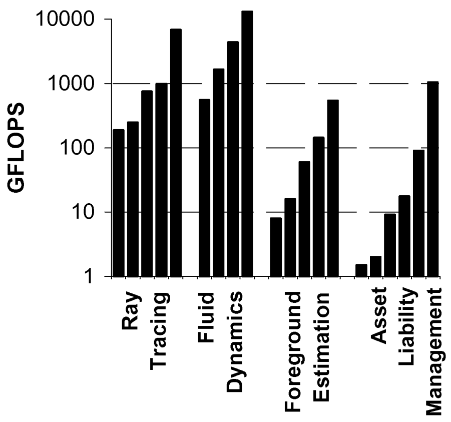


Fig. 17. ***Computation requirements of some RMS applications for a set of usage scenarios.***

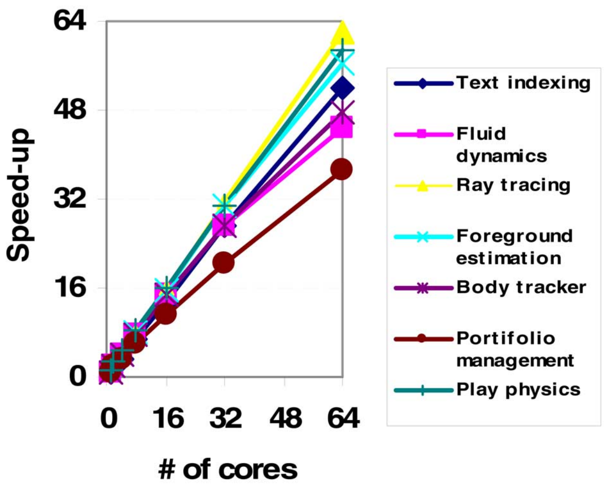


Fig. 18. ***Scalability of some RMS applications.***

However, achieving high performance from future systems is not as straightforward as this. There are a number of common challenges presented by RMS workloads from a system designer's point of view. In the rest of this section, we address some of the key challenges.

Programming Model: Parallelizing applications is a nontrivial task. Programmers must identify the parallelism in an algorithm and map or partition the computation to a large number of threads. In some cases, programmers must employ completely different algorithms than they would use in a single-threaded implementation because the single-threaded algorithm has too little inherent parallelism (or it is too difficult to identify or exploit). While there are existing languages and extensions to languages to facilitate this, they may prove too inaccessible for many programmers. Allowing a programmer to express certain parallel concepts through primitives, either through the programming language or a library, may reduce the effort of parallelization. Example concepts that would be useful to express are task dependence graphs, such as the elimination tree for a sparse solver, and reductions, which we discuss later.

Programmers also need to handle communication and synchronization between the threads. This is often done (in shared memory models) using locks. Keeping track of which locks protect what and avoiding deadlock by keeping the locks in a total order can be difficult even for a single programmer. For large code bases developed by many programmers simultaneously, this is a truly daunting task. However, this may be greatly simplified by techniques such as transactional memory [16], [17]. Transactional memory replaces critical sections protected by locks with "transactions." For example, a transaction may be delineated with "begin transaction" and "end transaction" instructions; the instructions within the transaction should happen atomically. With transactional memory, the programmer does not have to associate one or more synchronization variables with a critical section. This makes composing a large parallel application easier.

Fine-Grained Parallelism: Most current parallel systems were designed under the assumption that parallel sections (e.g., parallel loops) can be broken into large pieces (e.g., millions of instructions or more). However, a significant number of RMS workloads have key parallel sections that cannot be broken into large pieces. They have very fine-grained thread-level parallelism relative to today's parallel workloads. This is in part because in the iRMS context, problem sizes are often smaller than in the currently dominant parallel systems context of high-performance computing. In some iRMS cases, each chunk is only thousands or even hundreds of instructions. We observe this in various physical simulation, financial analytics, and computer vision workloads.

Furthermore, Amdahl's law tells us that the maximum parallel speedup for an application is bounded by the size of its serial sections. Therefore, in order to achieve good parallel speedups on a highly threaded architecture, even small modules in an application need to be parallelized.

For both of the above reasons, it is critical to provide efficient support for exploiting very fine-grained thread-level parallelism [19].

Data-Level Parallelism: In addition to thread-level parallelism, many RMS workloads have a lot of data-level parallelism. For example, computer vision workloads often include a significant amount of image processing, which is largely data-parallel. Also, many machine learning algorithms involve performing a set of arithmetic operations on each element of an input set. And linear solvers used in many domains include operations on vectors that are inherently data-parallel.

One of the most common and efficient methods for exploiting data-level parallelism is via single-instruction multiple-data (SIMD) execution, in which a single instruction operates on multiple data elements simultaneously. This is typically implemented by extending the width of registers and arithmetic logic units, allowing them to hold or operate on multiple data elements, respectively. The rest of the system is left untouched.

This implementation places some restrictions on the use of SIMD execution, particularly in the layout of the data. Operands and results are typically required to be grouped together sequentially in memory. To achieve the best performance, they should be placed into an address-aligned structure. If the data do not meet these layout restrictions, the programmer must convert the data's layout. This generally involves "scatter-gather" operations [11], where operands are gathered from multiple locations and packed into a tight group, and results are scattered from a tight grouping to multiple locations. Fig. 19 shows an example of a gather operation, where one field from

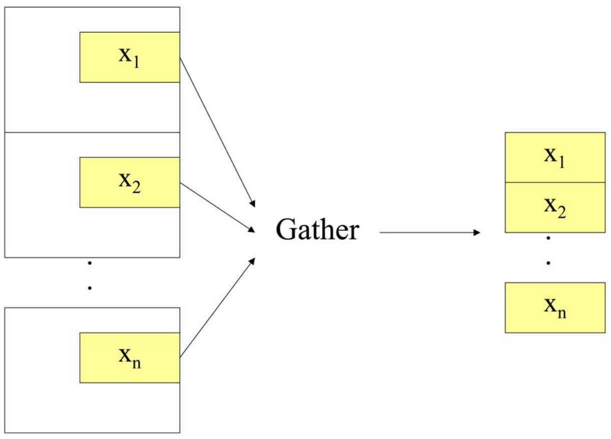


Fig. 19. ***Gather operation on an array of structures.***

each structure in an array of structures is read and packed into a contiguous stream. This can be quite expensive if done in software. Hardware acceleration of scatter-gather operations can enable significantly more RMS workloads to see benefits from SIMD execution, such as those that rely on sparse linear solvers.

An additional challenge imposed by this implementation of SIMD execution is control flow. Sometimes a programmer wishes to use conditional SIMD operations, where only a set of the data elements are operated on. This is typically supported through the use of bit masks. For each SIMD instruction, a bit mask can be provided to specify which operands to ignore. However, for a significant number of important RMS modules, such as various flavors of collision detection, the set of operands being masked out increases monotonically during execution. This leads to an increasingly large fraction of wasted effort by the system. If the hardware could support compressing the useful operands together, this inefficiency could be substantially reduced.

Memory Access Patterns: Many RMS workloads are dominated by streaming or repeated streaming memory access patterns. A streaming access pattern is one where an application touches a regular sequence of elements from a data structure. Each element is typically touched once, or, in some cases, multiple times in quick succession. This lack of reuse across a program's execution means that traditional caches provide little benefit. A repeated streaming pattern is one where the application touches the same sequence of elements multiple times; a cache can help here quite a bit, but only if it can hold the entire stream. Repeated streaming is seen, for example, in the conjugate gradient algorithm described earlier. For good performance, it is important for a system to efficiently handle these access patterns, both in terms of providing high bandwidth and in terms of providing low latency access to the data.

Memory bandwidth is a particular concern because on a multicore chip, it is shared by all of the threads on the chip. Unfortunately, the bandwidth requirements for some workloads we have examined are high even for a single thread because they access large data structures and perform little computation for each data item accessed. The data structures will most likely not fit in conventional caches, so each time an element is reused, it must be refetched from memory. Fig. 20 shows the memory bandwidth requirements for some RMS applications under different usage scenarios, assuming 16 MB of on-chip cache. For a number of applications and usage scenarios, the memory bandwidth requirement is in the hundreds of gigabytes per second, and in some cases exceeds 1 TB/s.

One possible solution to the memory bandwidth problem is to make caches much larger. Recent advances in multichip modules and 3-D stacking of silicon dies makes this feasible [4]. A second die containing an SRAM or even a high-density DRAM cache can be packaged with a processor to greatly increase the amount of storage in the processor package.

Architects have been working furiously for the past couple of decades to develop techniques to reduce or hide latency for accessing caches and memory. The most common technique for doing this is prefetching. However, existing prefetching techniques trade off higher bandwidth usage to achieve lower effective latency. Since bandwidth is in many cases more of a performance limiter than latency, more bandwidth-efficient prefetching techniques are needed. Further, most prefetching schemes assume streaming or repeated streaming patterns where the streams are long; the hardware learns the pattern and assumes it will continue. As discussed earlier, in many RMS workloads, the parallel tasks are small. An effect of this is that long streams are broken into small pieces (i.e., short streams). These short streams cannot be learned quickly enough by the hardware. Thus, software may need to provide some information to the hardware on what data will be accessed in the future.

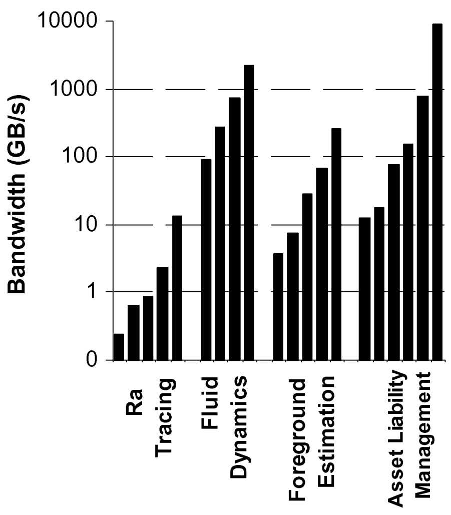


Fig. 20. ***Memory bandwidth requirements of some RMS applications for a set of usage scenarios.***

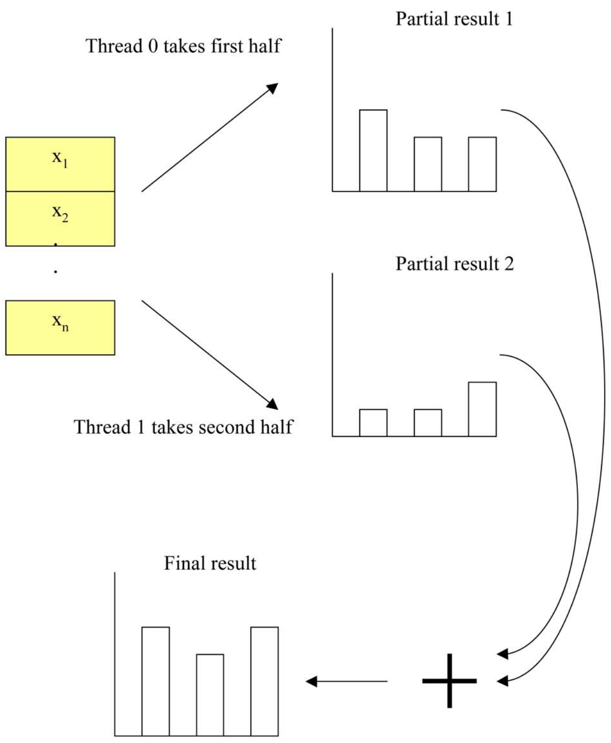


Fig. 21. ***Parallel histogram computation.***

Fast Atomic Operations: The most effective way to parallelize many key modules within RMS workloads is to have threads cooperate to produce a single result or a set of results. That is, rather than making a single thread responsible for each result, multiple threads may accumulate their partial results together. This is known as a parallel reduction. An example of a parallel reduction is the computation of a histogram from a set of input data, as shown in Fig. 21. One way to perform this computation in parallel is for each thread to compute a histogram for a subset of the data and then accumulate their partial results to a global total result. Operations similar to this are common in RMS workloads, such as in linear solvers.

For correctness, each reduction operation (e.g., addition to the global total result) must be atomic (i.e., protected with synchronization). Providing atomicity to protect reductions is quite expensive on current systems, and if reductions happen frequently, this can have a large impact on performance [12]. Support for fast atomic operations can significantly reduce these overheads and provide good performance benefits.

In some cases, the synchronization overhead is not the only performance problem for parallel reductions. If multiple threads attempt to accumulate results into the same accumulator simultaneously, they will serialize. If this happens often enough, this can significantly reduce the parallel speedup of an application. One way to alleviate this problem is to provide hardware support to 1) automatically allow each thread to perform its part of the reduction independently of the other threads (i.e., by providing a private copy of the accumulator) and 2) asynchronously accumulate each thread's partial result into the global result so that the threads can overlap useful work with the final stage of the reduction operation.

Fast Complex Arithmetic Operations: General-purpose processors are very fast at performing basic arithmetic operations (add, subtract, multiply, etc.) but much slower in performing more complex operations. Many RMS workloads, especially those from the finance domain and in machine learning and image processing, heavily use complex arithmetic operations such as transcendental operations and random number generation. Accelerating these operations could provide large performance benefits. One possible avenue for acceleration is special-purpose hardware. Another is to enable variable-precision operations. Current processors perform both simple and complex arithmetic operations with fixed precision. However, many modules do not need the full precision offered by the processor (e.g., image processing). Since complex arithmetic operations can often be sped up at the cost of some precision, enabling the programmer to make this precision–performance tradeoff may be beneficial.

V. CONCLUSIONS

In this paper, we described a general-purpose "analytics engine" that is at the core of next-generation processing platforms. The engine models events, objects, and concepts based on end-user input and accessible datasets. It iteratively refines its models in real-time. This engine is inspired by many emerging workloads, which are characterized along three fundamental classes of processing capabilities we call RMS—recognition, mining, and synthesis. We find that in addition to the analytics engine model, RMS workloads exhibit other fundamental similarities. They are based on a common set of mathematical models, numerical algorithms, and underlying data structures. We call this phenomenon workload convergence. Workload convergence has strong implications for future computer system developers because it means that a diverse set of emerging RMS applications from market segments like graphics, gaming, media-mining, unstructured information management, financial analytics, and interactive virtual communities presents a relatively focused, highly overlapping set of common platform challenges. A general-purpose processing platform designed to address these challenges has the potential to significantly enhance users' experience and programmer productivity. ■

REFERENCES

[1] G. Bebis *et al.*, "Advances in visual computing," in *Proc. 2nd Int. Symp., ISVC 2006*, Lake Tahoe, NV, Nov. 6–8, 2006.

[2] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities," *Sci. Amer.*, May 2001.

[3] T. Berners-Lee. (2003, Sep. 22). "WWW past and future," presented at the Royal Society, London, U.K. [Online]. Available: http://www.w3.org/2003/Talks/0922-rsoc-tbl/

[4] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb, "Die stacking (3D) microarchitecture," in *Proc. 39th Annu. Int. Symp. Microarchitect.*, 2006.

[5] F. Black and M. Scholes, "The pricing of options and corporate liabilities," *J. Political Econ.*, vol. 81, no. 3, pp. 637–654, 1973.

[6] S. Chakrabarti, *Mining the Web: Analysis of Hypertext and Semi Structured Data*, ser. Data Management Systems. San Mateo, CA: Morgan Kaufmann, 2003.

[7] G. Cornuejols and R. Tutuncu, "Optimization methods in finance," *Mathematics, Finance and Risk*, 2006.

[8] P. Dubey, "Recognition, mining and synthesis moves computers to the era of tera," *Technol.@Intel Mag.*, Feb. 2005.

[9] I. S. Duffi, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*. Oxford, U.K.: Clarendon, 1986.

[10] C. Ericson, *Realtime Collision Detection*, ser. Interactive 3D Technology. San Mateo, CA: Morgan Kaufmann, 2004.

[11] R. Espasa, F. Ardanaz, J. Emer, S. Felix, J. Gago, R. Gramunt, I. Hernandez, T. Juan, G. Lowney, M. Mattina, and A. Seznec, "Tarantula: A vector extension to the alpha architecture," in *Proc. 29th Annu. Int. Symp. Comput. Architect.*, 2002.

[12] M. J. Garzaran, M. Prvulovic, Y. Zhang, A. Jula, H. Yu, L. Rauchwerger, and J. Torrellas, "Architectural support for parallel reductions in scalable shared-memory multiprocessors," in *Proc. 10th Int. Conf. Parallel Architect. Compilation Tech.*, 2001.

[13] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE J. Robot. Automat.*, vol. 4, no. 2, pp. 193–203, 1988.

[14] P. Glasserman, *Monte Carlo Methods in Financial Engineering*. Berlin, Germany: Springer, 2003.

[15] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. Baltimore, MD, John Hopkins Univ. Press, 1996.

[16] M. Herlihy, J. Eliot, and B. Moss, "Transactional memory: Architectural support for lock-free data structures," in *Proc. 20th Annu. Int. Symp. Comput. Architect.*, May 1993.

[17] M. Herlihy, V. Luchangco, M. Moir, and W. N. Scherer, III, "Software transactional memory for dynamic-sized data structures," in *Proc. 22nd Annu. Symp. Principles Distrib. Comput.*, 2003.

[18] J. Hull, *Options, Futures and Other Derivatives*, 5th ed. Englewood Cliffs, NJ: Prentice-Hall, 2005.

[19] S. Kumar, C. J. Hughes, and A. Nguyen, "Carbon: Architectural support for fine-grained parallelism on chip multiprocessors," in *Proc. 34th Annu. Int. Symp. Comput. Architect.*, 2007.

[20] B. Liang and P. Dubey, "Recognition, mining and synthesis," *Intel Technol. J.*, vol. 9, May 2005.

[21] R. C. Merton, "Theory of rational option pricing," *Bell J. Econ. Manage. Sci.*, vol. 4, no. 1, pp. 141–183, 1973.

[22] E. Ng and B. Peyton, "Block sparse cholesky algorithms on advanced uniprocessor computers," *SIAM J. Sci. Comput.*, vol. 14, no. 5, pp. 1034–1056, 1993.

[23] D. A. Norman, *The Design of Everyday Things*. New York: Currency-Doubleday, 1989.

[24] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York: Springer-Verlag, 1999.

[25] M. Pharr and G. Humphreys, *Physically Based Rendering From Theory to Implementation*. San Mateo, CA: Morgan Kaufmann, 2004.

[26] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia, PA: SIAM, 2003.

ABOUT THE AUTHORS

Yen-Kuang Chen (Senior Member, IEEE) received the B.S. degree from National Taiwan University, Taiwan, and the Ph.D. degree from Princeton University, Princeton, NJ.

He is a Principal Engineer with the Corporate Technology Group, Intel Corporation, Santa Clara, CA. His research interests include developing innovative multimedia applications, studying the performance bottleneck in current architectures, and designing next-generation microprocessors/platforms. Currently, he is analyzing emerging multimedia applications and providing inputs to the definition of the next-generation CPUs and GPUs with many cores. He is one of the key contributors to Supplemental Streaming SIMD Extension 3 in the Intel Core 2 processor family.

Jatin Chhugani received the Ph.D. degree from The Johns Hopkins University, Baltimore, MD.

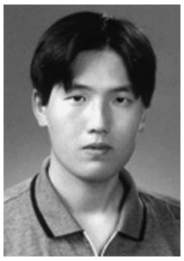
He is a Staff Researcher with the Corporate Technology Group, Intel Corporation, Santa Clara, CA. His research interests include developing algorithms for interactive computer graphics, parallel architectures, and image processing.

Pradeep Dubey (Fellow, IEEE) received the B.S. degree in electronics and communication engineering from Birla Institute of Technology, India, the M.S.E.E. degree from the University of Massachusetts at Amherst, and the Ph.D. degree in electrical engineering from Purdue University, West Lafayette, IN.

He is a Senior Principal Engineer and Manager of Innovative Platform Architecture with the Microprocessor Technology Labs, Corporate Technology Group, Intel Corporation, Santa Clara, CA. His research focus is computer architectures to efficiently handle new application paradigms for the future computing environment. He previously was with IBM's T. J. Watson Research Center and Broadcom Corporation. He was one of the principal architects of the AltiVec multimedia extension to Power PC architecture. He also worked on the design, architecture, and performance issues of various microprocessors, including Intel i386TM, i486TM, and Pentium processors. He has received more than 25 patents and has published extensively.

Christopher J. Hughes (Member, IEEE) received the B.S. and B.A. degrees from William March Rice University, Houston, TX, and the M.S. and Ph.D. degrees from the University of Illinois, Urbana-Champaign.

He is a Staff Researcher with the Microprocessor Technology Labs, Corporate Technology Group, Intel Corporation, Santa Clara, CA. His research interests include computer architecture and emerging workloads, with a current focus on workload-driven parallel architectures and memory hierarchies.



Daehyun Kim received the Ph.D. degree from Cornell University, Ithaca, NY.

He is a Senior Research Scientist with the Corporate Technology Group, Intel Corporation, Santa Clara, CA. His research interests include parallel computer architecture, intelligent memory systems, and emerging applications, focusing on workload-driven architecture design for large-scale chip multiprocessors.

Sanjeev Kumar (Member, IEEE) received the B.Tech. degree from the Indian Institute of Technology, Madras (now Chennai), the M.S. degree from Indiana University, Bloomington, and the Ph.D. degree from Princeton University, Princeton, NJ.

He is a Staff Researcher with the Microprocessor Technology Labs, Corporate Technology Group, Intel Corporation, Santa Clara, CA. His research interests include parallel architecture and software for emerging workloads.

Victor W. Lee received the S.M. degree from the Massachusetts Institute of Technology, Cambridge.

He is a Senior Staff Research Scientist with the Corporate Technology Group, Intel Corporation, Santa Clara, CA. His research interests are computer architecture and emerging workloads. He is currently involved in defining next generation chip-multiprocessor architectures.

Anthony D. Nguyen (Member, IEEE) received the Ph.D. degree from the University of Illinois, Urbana-Champaign.

He is a Senior Research Scientist with the Corporate Technology Group, Intel Corporation, Santa Clara, CA. His research interests include developing emerging applications for architecture research and designing the next-generation chip-multiprocessor systems.

Mikhail Smelyanskiy (Member, IEEE) received the Ph.D. degree from the University of Michigan, Ann Arbor.

He is a Senior Research Scientist with the Corporate Technology Group, Intel Corporation, Santa Clara, CA. His research focus is on building and analyzing parallel emerging workloads to drive the design of next-generation parallel architectures.