2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) | 978-1-6654-3577-2/20/$31.00 ©2021 IEEE | DOI: 10.1109/IPDPSW52791.2021.00160

IPDPS 2021 PhD Forum

To be held virtually on May 17, 2021
ipdps.org

Description

The IEEE Computer Society has created the Technical Consortium on High Performance Computing (TCHPC) to advance and coordinate work in the field of high performance computing networking, storage, and analysis concepts, technologies and applications, and to expand the IEEE's role in this interdisciplinary and pervasive field. The Consortium has launched an Education and Outreach Initiative to coordinate activities, information, and best practices around HPC education/outreach across its member technical committees and the broader community. This includes the coordination of student activities across conferences, and the lead chairs for this initiative also chair the IPDPS 2021 PhD Forum.

The annual IPDPS PhD Forum event was initiated in 2014 by the Technical Committee on Parallel Processing, the sponsor of the IPDPS conference series. It has grown from the traditional poster presentations by students working toward a PhD in broadly defined areas related to parallel and distributed processing to a broader, enhanced program to include sessions during the three days of the main conference to provide student participants coaching in scientific writing and presentation skills.

This year, it is being held virtually and will conduct a live session on Monday, May 17th with a lightning round of presentations by the 14 poster authors and a panel to discuss career paths with the PhD students who are participating in the Forum. Accepted posters will be hosted from the IPDPS 2021 virtual platform.

The usual poster presentations by students at a physical conference has been an opportunity to present their work and obtain feedback from the conference and to network with the other students and members of the community. This year to help enhance the virtual presentations, in addition to the online availability of the posters, students were invited to include an abstract describing their work in the proceedings of the conference workshops (IPDPSW 2021).

A compendium of these abstracts follows, noting that the posters will contain more detailed information including contact information of the authors. As the chairs for this event, we are pleased to present the work of these fourteen PhD candidates and hope that this 2021 experience will be a valuable enhancement to the work they are doing.

We also want to take this opportunity to thank the conference organizers for their support in holding this event, and we are especially grateful to the general chairs for their help and guidance throughout.

IPDPS 2021 PhD Forum Co-Chairs

Sanjukta Bhowmick (University of North Texas, USA)

Akshaye Dhawan (Bloomberg L.P., USA)

IPDPS 2021 PhD Forum

Research Projects Selected for Poster Presentation
May 17, 2021

1 *Resource Elasticity at Task-Level*
Jonas Posner, University of Kassel, Germany

2 *A CPU-GPU Scheduler Tolerant to Temporal Failures in Clouds*
Rafaela C. Brum, Federal Fluminense University, Brazil

3 *Evaluation of Vertex Reordering for Graph Applications*
Reet Barik, Washington State University, USA

4 *On the Predictability of Quantum Circuit Fidelity Using Machine Learning*
Norhan Elsayed Amer Abd Elgawad, Egypt-Japan University of Science and Technology (E-JUST)

5 *Improving the Operational Capability of Automated Empirical Performance Modeling*
Marcus Ritter, Technical University of Darmstadt, Germany

6 *Development of a Middleware to Create an Efficient Unified Programming Model for Heterogeneous Computing*
Pablo Antonio Martínez Sánchez, University of Murcia, Spain

7 *Task-Level Checkpointing for Nested Fork-Join Programs*
Lukas Reitz, University of Kassel, Germany

8 *E2Clab: Reproducible Analysis of Complex Workflows on the Edge-to-Cloud Continuum*
Daniel Rosendo, INRIA, France

9 *Verifiable Coded Computing: Towards Fast and Secure Distributed Computing*
Tingting Tang, University of Southern California, USA

10 *Task Scheduling in Reconfigurable Computing with OpenCL*
Pascal Jungblut, LMU Munich, Germany

11 *Hierarchical Cost Analysis for Distributed Deep Learning*
Haoran Wang, University of Orléans, France

12 *On the Road to a Unified Big Data and HPC Framework*
César Piñeiro-Pomar, Universidade de Santiago de Compostela (USC), Spain

13 *Pattern-Aware Vectorization for Sparse Matrix Computations*
Khaled Abdelaal, University of Oklahoma, USA

14 *Heterogeneity-Aware Deep Learning Workload Deployments on the Computing Continuum*
Thomas Bouvier, Inria Rennes, France

IPDPS 2021 PhD Forum
Poster #1 Abstract

Resource Elasticity at Task-Level

Jonas Posner
University of Kassel, PLM, Germany

Advisor: Prof. Dr. Claudia Fohry, University of Kassel, PLM, Germany

Abstract:

Adaptive resource management of supercomputers offers several benefits compared to conventional static resource management, including highly improved global throughput and decreased energy consumption. However, adaptivity must be backed by at least three major layers: global job schedulers, programming, and algorithms/applications. Recent research addresses these layers, but no comprehensive solution has yet been established.

Elastic algorithms cause a non-negligible additional development effort, and often focus on iterative approaches that automatically provide explicit synchronization points. This work, in contrast, proposes a novel resource elasticity scheme at the intermediate level of a task-based runtime system. Applications using our runtime automatically adapt to the addition and release of multiple compute nodes and dynamically balance the load without requiring explicit synchronization points or additional programming effort.

We build on the dynamic independent tasks pattern (DIT), which is well suited for tree-based algorithms solving search, optimization, and approximation problems. In DIT, tasks encapsulate sub-computations that can be executed in parallel to other tasks. Tasks cannot communicate with other tasks, except for parameter passing when generating child tasks at runtime. Typically, DIT is coupled with work stealing to balance the load dynamically at runtime. This work considers a multi-threaded lifeline-based variant in which each node runs a single process that maintains multiple worker threads. Local workers share tasks with other local workers, and only when the entire process runs out of tasks, the process attempts to steal tasks from random processes, followed by lifeline buddies. The latter are predetermined by a graph, and record unsuccessful steal requests and possibly answer them later.

In our elasticity scheme, addition and release of nodes is controlled by process 0, which can not be released. Resource changes can be triggered any time, but multiple requests are handled sequential. All resource change actions are performed in a distributed way and asynchronously to task processing, but not to work stealing. When releasing nodes, the corresponding processes stop processing tasks and send all remaining tasks and results to staying processes. In addition, the lifeline graph is recalculated to exclude the processes to be released from future work stealing. When adding nodes, new processes are started on these nodes, and then the lifeline graph is recalculated, with the new processes automatically being given tasks.

We implemented the elasticity scheme and conducted experiments with up to 128 nodes. We started one process per node, each with 40 worker threads, and analyzed running time overheads for halving and doubling the number of nodes, respectively. Releasing costs vary between 0.01s and 0.45s and are mainly caused by sending tasks and results from processes to be released to staying ones. Adding costs vary between 0.37s and 1.84s and are mainly caused by the delay of new workers until they can start task processing. Thus, adding nodes incurs slightly higher costs than releasing nodes. Since both releasing and adding new nodes is performed in a distributed way and asynchronously to task processing, the costs increase only gently with the number of nodes, resulting in good scalability.

IPDPS 2021 PhD Forum
Poster #2 Abstract

A CPU-GPU Scheduler Tolerant to Temporal Failures in Clouds

Rafaela Correia Brum
Federal Fluminense University, Brazil

Advisors: Lúcia M. A. Drummond (Federal Fluminense University) and
Maria Clicia S. de Castro (State University of Rio de Janeiro)

Abstract:

Cloud computing has emerged as a practical and cheap way of executing applications as the users do not need to pay for the servers' maintenance or the energy consumed. One of the most common resources in a cloud environment is a virtual machine (VM). Cloud providers offer several types of VMs in different markets with different availability guarantees. Amazon EC2 presents two main markets to deploy a VM: the On-demand and the Spot markets. The On-demand market allocates VMs for a fixed cost per time unit, and they remain available during the whole execution. On the other hand, the Spot market offers VMs with a steep price discount, but EC2 can terminate the instance at any time.

In this work, we provide a cloud scheduler for CPU-GPU applications using Spot instances as much as possible, respecting the application deadline constraints while minimizing the monetary costs. As this is a work in progress, the first scheduler version only supports applications running in a single VM with one GPU. Before starting the execution, the scheduler executes a greedy algorithm to select the initial Spot GPU VM. The algorithm searches for the instance type that meets the deadline and presents the minimum monetary cost.

When a revocation occurs, the scheduler tries to allocate the application on a different instance type of the Spot market. The scheduler recalculates both the expected finishing time and the cost considering both the percentage of the application already computed and the total time elapsed since the beginning of the scheduler execution. If no suitable Spot instance is available, the scheduler selects an on-demand instance type.

To evaluate the framework, we used a fault-tolerant and single-GPU version of a sequence alignment algorithm named MASA-CUDAlign. We executed the application with five different DNA pairs of sequences, whose sizes varied from 27 to 66 million base pairs (MBP). Spot and On-demand VM prices were obtained on November 3, 2020, in the us-east-1 EC2 availability zone.

In order to measure the impact of our framework on the execution time, we executed the first experiment in the g4dn.xlarge instance. The framework overhead observed was less than 3% on average. In the second experiment set, the scheduler could choose among five instance types: g2.2xlarge, g3s.xlarge, g4dn.xlarge, g4dn.2xlarge, and p2.xlarge. We simulated several revocation scenarios, following the Poisson model, and observed the impact on the total execution time and the monetary costs compared to the execution in a g4dn.xlarge on-demand instance. In these experiments, we observed a 60% cost reduction with a corresponding 13% execution time increase, on average, which indicated that the use of Spot instances in fault-tolerant applications can be promising even in scenarios with several revocations.

IPDPS 2021 PhD Forum
Poster #3 Abstract

Evaluation of Vertex Reordering for Graph Applications

Reet Barik
Washington State University, USA

Advisor: Ananth Kalyanaraman (WSU), **Co-Advisor:** Mahantesh Halappanavar (PNNL)
Co-authors for other parts of the research: Marco Minutoli (PNNL), Nathan R. Tallent (PNNL)

Abstract:

Data movement has been the primary bottleneck for nearly all graph-theoretic applications. Irregular memory accesses that characterize most graph algorithms make preserving locality difficult. This is compounded by the scale-free characteristics of real-world graphs which generally follow the power-law degree distribution. The intuition behind vertex (re)ordering to improve locality is as follows: given an input (or natural) order of vertices, compute a permutation of the order such that proximity between pairs of vertices in the graph-space is reflected in the proximity of their labels or "ranks". This increases the probability of finding the data contained in the neighbors of a vertex in the same cache line and hence leads to lower data movement thereby improving performance. To improve locality in computation, several vertex (re)ordering schemes have been proposed. However, a detailed characterization (both empirical and analytical) of various reordering schemes or their impacts on end applications has been largely missing in the literature.

Problem: In the study (published at the IISWC'20 conference) that this poster is based on, we present an extensive empirical evaluation of up to 11 reordering schemes, taken from different classes of approaches, on a set of 34 real-world graphs emerging from different application domains. Our study is presented in two parts: a) a thorough comparative evaluation of the different schemes on their effectiveness to optimize a set of linear arrangement gap measures, relevant to preserving locality; and b) an extensive evaluation of the impact of the ordering schemes on two real-world, parallel graph applications, namely, community detection and influence maximization.

Performance based on gap measures: Our studies show a significant divergence among the ordering schemes (up to 40x between the best and the poor) in their effectiveness to reduce the gap measures. Partition-based techniques like METIS, Grappolo, and Rabbit-Order outperform others in minimizing the average gap profile, with Reverse Cuthill-McKee showing competitive performance as well. In fact, we can observe four distinct performance tiers of schemes. The top-performing group is constituted by METIS-32, Grappolo, and Rabbit-Order; followed by RCM which generates an average gap profile that is between roughly 1x–8x more than the first group for at least 50% of the inputs. The third group consisting of a mixture of schemes from different categories generates an average gap profile that is roughly between 5x–25x larger, and the final group constituting of degree-/hub-based schemes is roughly between 10x–40x larger. We conclude that the partition-based schemes and RCM are superior to other schemes by this edge gap statistic.

Application impact: The performance of four different schemes was evaluated on a parallel community detection implementation. The key takeaway is that Grappolo usually outperforms Degree Sort, at times by factors 2x–4x when looking at Phase and Iteration times. It also usually has the highest Parallel Efficiency (Work%). It also has the lowest work per edge resulting in better load balancing. As for influence maximization, our evaluations showed little separation among the ordering schemes. This is owing to the application characteristic which performs many probabilistic BFS's.

IPDPS 2021 PhD Forum
Poster #4 Abstract

On the Predictability of Quantum Circuit Fidelity Using Machine Learning

Norhan Elsayed Amer Abd Elgawad
Egypt-Japan University of Science and Technology (E-JUST)

Advisor: Ahmed El Mahdy, Egypt-Japan University of Science and Technology (E-JUST)

Abstract:

Quantum computing is an open research area that is quickly developing to reach practical quantum computing systems eventually. However, the limited number of qubits in the Noisy Intermediate-Scale Quantum era leads to errors that can alter qubits' measurements. These errors occur due to multiple reasons. Several researchers have optimized quantum compilers to mitigate errors. Some researchers have contributed to mitigating cross-talk errors that occur when multiple gates are executed simultaneously. Other researchers have proposed schemes to map qubits according to layout requirements to reduce adding swap gates that would increase error. Other research works have considered decreasing the number of swap gates, eventually decreasing the circuit depth, leading to a decrease in error. However, the available error mitigation methods are still developing in quantum compilers considering different error type at a time. One important research question is whether we can apply classical optimization techniques, including machine learning to quantum compilers, referred to as "classical optimizations under quantum constraints". This has the potential for modeling more complex interactions among gates and qubits. To tackle this question, we start by exploring the predictability of circuit fidelity through machine learning.

In particular, we utilize IBM's Qiskit software to generate random quantum circuits, provide the corresponding circuit state vector, and calculate the fidelity distance between the output of the state tomography of a circuit and its state vector. The generated random circuits consist of different combinations of basis gates (ID, RZ, SX, and X) of 5 stages depth thresholds. These circuits are then fed to our deep learning model to infer corresponding fidelities. Specifically, we employ an embedding layer in our model to encode a dense representation of a circuit which is then fed to multiple dense layers. We optimize our model using RMSprop with a learning rate of 0.0009 that minimizes the mean absolute error. Our model can be then used to optimize the allocation of logical qubits and gates to physical ones by choosing the best-predicted circuit out of the circuit design space for a given target state. To validate our approach, we calculate the correlation factor between the predicted fidelity and the ground truth. Our preliminary results on single-qubit machines have achieved a correlation factor of 0.806. In addition, there is a high correlation of 0.907 and a mean absolute error of 0.005 between the ground truth fidelity of the circuit chosen by our model and the maximum ground truth fidelity of the circuit available in the limited design space for a given state vector. We plan to extend this work to five qubit machines and explore reinforcement learning to further optimize quantum circuits.

IPDPS 2021 PhD Forum
Poster #5 Abstract

Improving the Operational Capability of Automated Empirical Performance Modeling

Marcus Ritter
Department of Computer Science, Technical University of Darmstadt, Germany

Advisor: Felix Wolf, Department of Computer Science, Technical University of Darmstadt, Germany
Co-authors: Alexandru Calotoiu, Alexander Geiß, Thorsten Reimann, Torsten Hoefler, Sebastian Rinke

Abstract:

The design and development of HPC applications is an extremely challenging task, requiring major resource investments for them to run efficiently on existing and future large-scale machines. As the demand for parallelism and HPC is constantly increasing, so is the need for performance analysis. Empirical performance modeling is a powerful tool to analyze the performance behavior of parallel applications. In general, a performance model provides an analytical expression of an application's behavior at different scales. However, identifying scalability bottlenecks in parallel applications is a vital but also laborious and expensive task. Empirical performance modeling techniques have proven useful in finding such constraints, although there are often limitations, such as the modeling budget or the amount of noise on the required performance measurements, that severely limit model accuracy. Extra-P is such an empirical modeling tool that automatically generates human-readable performance models from performance data. To create a model Extra-P requires a set of small-scale experiments using different combinations of the application's execution parameters. One constraint of Extra-P is the budget required for conducting the necessary performance measurements. Currently, Extra-P requires e = 5^(m+1) performance experiments$e, where m is the number of parameters, therefore requiring 625 experiments to model an application with three configuration parameters, which is usually too expensive to be practical. Another constraint limiting the operational capability of Extra-P is the amount of noise on the conducted performance measurements. Similar to other regression-based empirical modeling techniques, the accuracy of the created models decreases with an increasing amount of noise, as it becomes more and more difficult to distinguish signal from noise, especially when dealing with multiple application parameters. We present two methods, implemented and integrated into Extra-P, to overcome these restrictions: The first method identifies a novel parameter-value selection heuristic, which functions as a guideline for the experiment design of the performance measurements, leveraging sparse performance-modeling, a technique that requires only a fraction of the usual modeling budget. The second method exploits the ability of deep neural networks to discover the effects of numerical parameters, such as the number of processes or the problem size, on performance when dealing with noisy measurements, to increase noise resilience. We evaluate both methods using a combination of synthetic analysis and data from three different case studies, to compare their accuracy, predictive power, modeling cost, and computational overhead with the base performance of Extra-P. The Sparse performance-modeling approach in combination with our efficient heuristic parameter-value selection strategy successfully reduces the average modeling cost by about 85% while retaining 92% of the model accuracy, additionally allowing more flexibility in selecting the parameter values for the required performance measurements, enabling model creation in some scenarios. Our adaptive modeling approach on the other hand effectively reduces the impact noise often has on the creation of empirical models describing the effects of numerical parameters, such as the number of processes or the problem size, on performance significantly improving model accuracy by up to 25% and predictive power by about 15%.

IPDPS 2021 PhD Forum
Poster #6 Abstract

Development of a Middleware to Create an Efficient Unified Programming Model for Heterogeneous Computing

Pablo Antonio Martínez Sánchez
University of Murcia, Spain

Advisors: José Manuel García, Gregorio Bernabé

Abstract:

More and more often it is said that the glorious era of the CPUs is ending. The decline of the CPUs is caused by the dramatic reduction in the performance improvements in the lasts years,forgetting Moore's law and Dennard scaling. The future of computer architecture seems to be heterogeneous. In a heterogeneous environment, multiple specialized exist (accelerators), and each one is specialized in certain tasks. Nowadays, the most powerful computers include GPGPU, instead of concentrating all of its power in CPUs. Big companies are looking forward to heterogeneous hardware; Microsoft and Amazon use FPGAs in the cloud, Google work with TPUs and Apple recently announced Apple M1.

However, the adoption of these ideas in the software area is still in its infancy; each of these devices must be programmed using different languages and technologies. It makes software development much harder. Years ago appeared the idea of Single Source Multiple Devices (SSMD), a language that provides a unique interface to program many heterogeneous devices. Some modern examples are; oneAPI, PHAST or HPVM. Nonetheless, these languages are rarely capable of achieving a good performance in all the hardware they support. This problem is known as performance portability.

Another important aspect is to make software capable of exploiting efficiently all the hardware resources. In other words, if a heterogeneous system has a CPU, GPU, and some accelerator for a given task, the software should distribute the workload between all the devices automatically. This would dramatically reduce the execution time and/or energy consumption. To divide the workload between devices, a scheduling is needed.

Modern SSMD languages often rely on LLVM, an infrastructure that allows easy development of compilers. It also comes with a key idea in modern compilers; the intermediate representation (IR), which is an intermediate language that is between the source code and the assembly language. This allows uncountable opportunities for enhancements and optimizations. MLIR is another compiler structure based on LLVM that provides a much richer solution in terms of IR, which allows solving different issues in LLVM.

This thesis aims to explore the different problems inside the growing heterogeneous computing world, proposing solutions to them. Specifically, we have proven in previous work that performance portability is possible with enough effort of the programmer and the SSMD developers. Then we aim to propose a solution to distribute a given workload between all the available devices in a system automatically. After developing a theoretical model, we plan to implement these ideas as a middleware inside an SSMD or to create an LLVM/MLIR solution. This middleware will attack the performance portability from another point of view: instead of ensuring that all of the devices provide good performance for all the workload, select the best of the devices for each task.

IPDPS 2021 PhD Forum
Poster #7 Abstract

Task-Level Checkpointing for Nested Fork-Join Programs

Lukas Reitz
Research Group Programming Languages / Methodologies
University of Kassel, Germany

Advisor: Prof. Dr. Claudia Fohry, University of Kassel

Abstract:

Fault tolerance is often realized through checkpointing and, upon failure, continuing the application from a previously saved checkpoint. It is is an established technique for tolerating fail-stop failures in distributed applications. Recently, task-level checkpointing has received some attention. It achieves efficiency through saving only task descriptors to a resilient store.

We consider the nested fork-join (NFJ) task model, where tasks may spawn several child tasks, wait for their children's results, and return a result to their parent. Tasks are usually represented by stack frames. The NFJ model is suitable for, e.g., divide-and-conquer algorithms. NFJ programs are often implemented with work stealing, where processes, called thieves, steal tasks from other processes, called victims. An example of an NFJ programming environment is Cilk.

Recently, a task-level checkpointing scheme for the NFJ model has been sketched. It saves checkpoints at regular time intervals, in the event of work stealing, and during recovery. Upon failure, it recovers locally, confining the failure handling to a small subset of the processes. Moreover, it deploys shrinking recovery, i.e., the program execution continues on a reduced set of processes. The original scheme has been designed for programs that restrict the number of worker threads per process to one. On computing nodes equipped with several processing units, however, it is often beneficial to run a small number of processes with multiple worker threads each.

This work extends the original scheme to support such cases. We assume that each worker thread maintains an own task queue. Moreover, the workers of a process use shared data, for instance intermediate results. Checkpoints are saved per process instead of per worker, as this considerably reduces the overall number of checkpoints. The original scheme defines the contents of checkpoints for single worker processes. We modify their definition to include the task queues of all workers as well as the shared data.

The process of constructing a valid checkpoint and writing it to a resilient store, henceforth called backup writing, requires consideration of all work stealing-related operations that are possibly performed during this process. A naive approach to backup writing could synchronize and pause all workers of a process from the beginning until the end of backup writing. We devised a more efficient alternative, in which the workers first construct a checkpoint via independent contributions, and then one of them writes it to a resilient store. This way, workers need not pause.

Backup writing of a process is initiated by an arbitrary worker, e.g., after some fixed time interval. Each worker checks in its communication phase, whether a backup writing has begun, and if so, it copies its task queue. The last worker additionally writes the task queue copies and the shared data to a resilient store.

We expect the extended scheme to perform similar to the original one. Future work should implement and experimentally evaluate it.

IPDPS 2021 PhD Forum
Poster #8 Abstract

E2Clab: Reproducible Analysis of Complex Workflows on the Edge-to-Cloud Continuum

Daniel Rosendo
INRIA, France

Advisor: Gabriel Antoniu, INRIA

Abstract:

Distributed digital infrastructures for computation and analytics are now evolving towards an interconnected ecosystem allowing complex applications to be executed from IoT Edge devices to the HPC Cloud (aka the Computing Continuum, the Digital Continuum, or the Transcontinuum). Understanding end-to-end performance in such a complex continuum is challenging. This breaks down to reconciling many, typically contradicting application requirements and constraints with low-level infrastructure design choices. One important challenge is to accurately reproduce relevant behaviors of a given application workflow and representative settings of the physical infrastructure underlying this complex continuum. We introduce a rigorous methodology for such a process and validate it through E2Clab. It is the first platform to support the complete experimental cycle across the Computing Continuum: deployment, analysis, optimization. Preliminary results with real-life use cases show that E2Clab allows one to understand and improve performance, by correlating it to the parameter settings, the resource usage and the specifics of the underlying infrastructure.

IPDPS 2021 PhD Forum
Poster #9 Abstract

Verifiable Coded Computing: Towards Fast and Secure Distributed Computing

Tingting Tang
University of Southern California

Advisor: Murali Annavaram, University of Southern California
Co-authors: Hanieh Hashemi, Ramy E. Ali, Tynan Gangwani, and Salman Avestimehr

Abstract:

Distributed computing using cloud resources is being widely used as it allows users to offload their compute-intensive tasks to run on multiple cloud servers, and thus enable the execution of many large-scale applications, such as machine learning. Execution speed variations are commonly observed among compute nodes by up to an order of magnitude, caused by multiple factors such as the heterogeneous setting of machines, resource contention of shared servers, disk IO delay, and hardware faults. These factors significantly degrade the performance when a single slow machine (called a straggler) can create bottlenecks to the whole distributed computing. Apart from stragglers, attackers who compromise cloud resources, or untrusted data center operators may lead to substantial concerns regarding whether the computation performed by a cloud server can be trustworthy. Moreover, even a trusted cloud may return inaccurate results unintentionally. Coded computing techniques inject computational redundancy in coded format to mitigate stragglers and handle the computational integrity. The goal in coded computing is to compute a multivariate polynomial $(Xi), \forall i \in [K]$over a dataset X=(XT1,XT2,⋯,XTK)T. For example, the Lagrange coded computing (LCC) framework encodes this data set into Ncoded datasets X'1, X'2,⋯, X'N,where $N$is the number of nodes and requires NLCC to be at least $(K-1)$ deg $f+S+2M+1$ to tolerate S stragglers and M malicious nodes. That is, malicious nodes are twice as costly as stragglers in LCC. The overall computation results in LCC can be recovered when at least NLCC$-S$ nodes return their computations. Another line of research uses verifiable computations to provide information-theoretic guarantee to the integrity of computation outsourced to an untrusted server. We propose Verifiable Coded Computing (VCC), a unified framework to simultaneously mitigate stragglers and provide security against Byzantine nodes, with a lower cost of tolerating malicious nodes than LCC. Compared to LCC, the minimum number of nodes NVCC required by VCC is only $(K-1)$ deg $f+S+M+1$, and the cost of a malicious node is the same as a straggler node in VCC. The key idea of VCC is to encode the input dataset to create computational redundancy in a coded form across the nodes. This redundancy can then be exploited to mitigate straggler effect. The second part of VCC ensures security by verifying the computation of each node independently using node's own compute results with a method similar to Freivalds' algorithm until we get the minimum number of verified results required for decoding. This verification step can start as soon as the first node responds, unlike LCC which requires NLCC$-S$to respond before starting to decode. The third part of VCC is a dynamic coding scheme that updates the coding configuration when malicious nodes are detected and removed from the pool of available nodes. Depending on the presence of stragglers, different strategies towards straggler tolerance may be chosen to change the coding.

IPDPS 2021 PhD Forum
Poster #10 Abstract

Task Scheduling in Reconfigurable Computing with OpenCL

Pascal Jungblut
LMU Munich, Germany

Advisor: Prof. Dieter Kranzlmüller, LMU Munich, Germany

Abstract:

FPGAs have been a promising energy-efficient extension to the existing hardware landscape in HPC. Even though the programmability made significant progress with the commercial availability of High Level Synthesis (HLS), the adoption is slow. With our work we want to lay the scientific foundation for further, usable and pragmatic programming abstractions for reconfigurable hardware. We focus on reconfiguration-aware scheduling strategies for task execution on FPGAs.

Our main research question is: how to systematically evaluate scheduling strategies for HLS programs on FPGAs? Existing approaches use partial reconfiguration (PR) to schedule tasks on the FPGA substrate. While this gives the runtime fine grained control over the loaded configuration (bitstream) on the chip, it leads to considerable overhead during the development and incompatibilities with other implementations. Instead of PR we consider entire configurations that contain weighted implementations for multiple tasks. An advantage is that we do not make any assumptions about the reconfigurable system, because we can use models for the task execution and workload on FPGAs and project the execution strategies based on the model(s).

The task performance is modeled using an α-β-model. Preliminary results show high accuracy for compute kernels. The parameters α and β can be obtained offline (i.e. before execution starts), online or in a hybrid mode using a simple linear regression. The latter is especially useful since α can be seen as constant across all kernels and may thus be evaluated once offline.

We developed a platform-independent scheduler on top of OpenCL that can either determine the values for α and β during execution or use input from the offline models. Early results show that the approach can select the optimal configuration and schedule the tasks accordingly.

In the future we want to generate more workload models for typical scientific applications and evaluate the scheduling strategies using the models.

IPDPS 2021 PhD Forum
Poster #11 Abstract

Hierarchical Cost Analysis for Distributed DL

Haoran Wang
LIFO - Université d'Orléans and Huawei Paris Research Center, France

Advisors: Sébastien Limet and Sophie Robert, LIFO - Université d'Orléans;
Chong Li, Huawei Paris Research Center

Abstract:

Deep Learning (DL) developed rapidly during the past decade. DNN models become larger and more complex. Increasing size of datasets and models, requires efficient distributed approaches. Different parallelism strategies result in different performance depending on the structure of DNNs. In order to obtain better performance and overcome the memory restrictions, *Hybrid Parallelisms* (HP), which apply different basic parallelism strategies on different parts of DNNs, are also explored. However, different parallelisms bring about mixed extra costs which are difficult to distinguish and evaluate. It is crucial to provide an approach which could clearly evaluate the costs caused by parallelisms and systematically find efficient hybrid strategies. Current approaches only consider one or two kinds of parallelisms.

In this work, we firstly present the training process of DNNs and give the explanations of the DNN notions. Three basic parallelism strategies (DP: data parallelism, OP: operator parallelism, PP: pipeline parallelism) are introduced and their relative merits are compared. The computation and communication are naturally distinguished when only considering the HP of DP/PP. DP determines the synchronous communication, but has no effect on forward/backward propagation (FPG/BPG), PP causes a bubble and little communication but will not affect the synchronization. Fundamental cost analysis can be easily applied. However, when taking OP into consideration, the FPG/BPG are changed to a mixed process of communication and computation where the previous cost analysis is no longer suitable. Existing approaches have not created a concrete execution model for distributed DNN training, but simply evaluate the total cost of computation and communication. Without distinguishing different kinds of communication, the proper HP can only be obtained through traversal or tuning, and loss further optimization opportunities.

In order to formalize the behaviors of the HP in distributed DL and quantitatively evaluate the cost caused by HP, we are studying Bridging DL composed by a double-level execution model associated with a symbolic cost model. The double-level execution model is used to explore the details of the HPs. The training process of the whole DNN model is abstracted by a *super-step* while the training of an operator is abstracted by a *micro-step*. With the two *steps*, the mixed communication and computation caused by three basic parallelism strategies are properly separated and the training process is clearly described. Based on the double-level execution model, the cost model of distributed DL helps to choose efficient HP strategies. Details of the cost model can be found in the poster and extended abstracts.

To conclude, Bridging DL enables systematical HP searching for distributed DNN training and provides a good opportunity for further analyzing and optimizations. Besides, Bridging DL also helps to guide the DNN framework for code generation of HP. A preliminary model (for micro-step) has been implemented on MindSpore. The following parts will be implemented and evaluated.

IPDPS 2021 PhD Forum
Poster #12 Abstract

On the road to a unified Big Data and HPC framework

César Piñeiro-Pomar
CiTIUS, Universidade de Santiago de Compostela, Spain

Advisor: Juan C. Pichel, CiTIUS, Universidade de Santiago de Compostela

Abstract:

Nowadays we are living in the Big Data era. The de facto standards for parallel processing of Big Data are Apache Hadoop and Apache Spark engines. These frameworks require implement their applications in Python, Java or Scala using programming paradigms such as MapReduce. However, HPC applications have always been implemented in Fortran and C/C++ in order to exploit multithreading (OpenMP) and multiprocessing (MPI) capabilities of clusters. As a consequence, this interoperability divergence between HPC and Big Data languages and programming models difficulties the creation of applications that bring together the advantages of HPC and Big data worlds.

To deal with that issue we introduce Ignis, a new Big Data-HPC framework that allows the execution of applications that combine multiple programming languages without additional overhead. Our framework uses a multi-language RPC approach to create a native executor for each language and a modular design pattern that facilitates the inclusion of new languages. All Ignis communications are internally implemented using MPI collective operations, which allows users to execute MPI native applications on an Ignis cluster. In this way, MPI and Spark-like codes can be combined in the same application. Unlike previous works, our proposal is a step forward in the convergence of HPC and Big Data since applications belonging to both worlds can be executed efficiently in the same framework. Note that Ignis is fully developed and executed inside Docker containers, which avoids any system dependency and library incompatibility issues.

Our architecture is divided into four independent main modules: Submitter, Backend, Driver and Executor. They are coded in different languages, using Apache Thrift over an SSH tunnel for the inter-module communications. The submitter is a simple script similar to spark-submit that calls a cluster manager (Mesos) to create a driver instance. It lives in the Submitter container where users prepare the environment before launching a job. The Executor contains a collection of functions that the Backend uses to implement its logic. Each language has its own implementation and can communicate with each other to share data. The Driver is a user API through which users can access all the available functionalities of the framework. This module does not perform any heavy computation and uses Thrift RPC to delegate its work to the Backend that implement the Driver functionality.

Performance tests were carried out sorting in ascending order 1TB of text data, which contains about two billion lines. We have compared the sort built-in capability of Spark with Ignis using a C++ executor and a Python driver code. Ignis Terasort is between 2.06x and 1.45x faster than Spark using from 4 to 10 nodes with 32 cores per node. Regarding to the memory consumption, Spark has a penalization of extra 40 bytes per string (for each of the lines in the input data). Moreover, it requires 128MB for each JVM. On the other hand, a C++ application only needs 24 extra bytes per string. It means that, ignoring the JVMs overhead, Spark will always consume 1.6x more memory than the Ignis C++ executor.

IPDPS 2021 PhD Forum
Poster #13 Abstract

Pattern-Aware Vectorization for Sparse Matrix Computations

Khaled Abdelaal
University of Oklahoma, USA

Co-authors: Richard Veras and Martin Kong

Abstract:

Sparse data is being used in a variety of High-Performance applications such as Big Data Analytics, Social Networks, Scientific Computing, and Machine Learning. These applications perform computations on sparse data such as sparse-matrix dense vector multiplication, sparse matrix-dense matrix multiplication, among many others.

To accelerate applications that make use of sparse data, several specialized storage and computation algorithms have been introduced. Existing work suffers from numerous limitations. Two of the main ones are: i) the semantic gap between sparse storage formats and the optimizations/transformations applicable on them and ii) the relatively poor support of general-purpose compilers to optimize sparse-computations. For example, current support of back-end compilers (e.g., GCC) for automatic sparse-vectorization is, to our surprise, very limited, failing to recognize vectorizable code for several simple regular, but sparse patterns. Also, the introduction of efficient storage formats alone for sparse data does not exploit the full potential of optimizations since they do not automatically extract dense segments in the sparse structures. They also introduce an additional layer of complexity to computations on some of these formats.

In this work-in-progress, we show the potential of applying vectorization by analyzing three graphs from SNAP dataset: facebook, twitter, and hepTh. Patterns (sequences of values) of different vector sizes (4, 8, 16, and 32) were explored. Our analysis shows that focusing on a few patterns covers most of the non-zeros within the matrix, indicating the potential for vectorization. We also explore a set of techniques to enable fast and low-overhead computations on sparse matrices. First, our framework-in-progress extracts metadata about dense segments in the sparse structure. This can be achieved using an efficient scanning algorithm that employs hierarchical representations of the sparse structures. The next stage is to generate vector code to exploit data-level parallelism. Having the metadata from the first stage describing the location of dense regions facilitates better exploitation of vector resources. This information can be leveraged together with Gather-Scatter operations that allow to pack and combine operations from different dense regions. Finally, we are exploring different techniques to generate vector codelets tuned to each vector pattern, including determining the capabilities of back-end compilers (GCC, Clang, ICC) to handle sparse and predicated vector computations, synthesizing our own SIMD-vector intrinsics, and using polyhedral-compilation techniques to systematically combine vector patterns.

We evaluated some of our ideas on a matrix-vector multiplication using the Facebook graph from the SNAP dataset, a 4039x4039 matrix. Our approach first extracts the vector metadata representing dense computational regions; The sparse matrix-vector multiplication is then performed with AVX256 vector intrinsics (vector size of 8 32bit floats). We compare our proposed technique against 3 different variants: i) a dense naïve implementation that neither uses vectorization nor an efficient storage format; ii) a variant that relies on Clang's support to automatically generate vector code; and iii) a third variant using sparse coordinate format (COO). Our preliminary results show the high potential of combining efficient storage, dense segments extraction, and vectorization to accelerate sparse data computations.

IPDPS 2021 PhD Forum
Poster #14 Abstract

Heterogeneity-aware Deep Learning Workload Deployments on the Computing Continuum

Thomas Bouvier
Univ Rennes, INSA, Inria, CNRS, IRISA — Rennes, France

Advisors: Alexandru Costan and Gabriel Antoniu,
Univ Rennes, INSA, Inria, CNRS, IRISA — Rennes, France

Abstract:

The increasing need for real-time analytics motivated the emergence of new incremental methods to learn representations from continuous flows of data, especially in the context of the Internet of Things. With an ever-growing amount of input data being generated by sensors, this trend led to the evolution of centralized computing infrastructures towards interconnected processing units spanning from edge devices to cloud data centers. Such infrastructure aims to benefit from data locality to decrease communication costs and latency. This new paradigm is referred to as the Computing or Edge-to-Cloud Continuum.

However, the network and compute heterogeneity across and within clusters may negatively impact Deep Learning (DL) workloads, leading to stragglers slowing down the whole training process. Conciliating these constraints in large-scale scenarios is therefore a problem gaining more and more interest from both the DL and the HPC – Big Data communities.

In this work, we introduce a roadmap for understanding the end-to-end performance of DL workloads in such heterogeneous settings. The goal is to identify key parameters leading to infrastructure bottlenecks and devise novel intra- and inter- cluster strategies to address them. In particular, we distinguish deterministic heterogeneity, caused by hardware imbalances (i.e., different compute resources, slow network links), from dynamic heterogeneity, caused by unexpected events occurring during the execution (i.e., I/O bottlenecks, temporary network slowdowns). This approach allows for a better distribution of the initial workload by considering the actual performance of nodes, and then mitigates the disruptions at runtime by using asynchronous gradient descent algorithms. We will explore various policies aiming to improve makespan, cost and fairness objectives while ensuring system scalability.