ReDSEa: Automated Acceleration of Triangular Solver on Supercloud Heterogeneous Systems

Georgios Zacharopoulos
Computing Systems Lab
Huawei
Zurich Research Center
Zürich, Switzerland

Ilias Bournias
Computing Systems Lab
Huawei
Zurich Research Center
Zürich, Switzerland

Verner Vlačić
Computing Systems Lab
Huawei
Zurich Research Center
Zürich, Switzerland

Lukas Cavigelli
Computing Systems Lab
Huawei
Zurich Research Center
Zürich, Switzerland

***Abstract*—When utilized effectively, Supercloud heterogeneous systems have the potential to significantly enhance performance. Our ReDSEa tool-chain automates the mapping, load balancing, scheduling, parallelism, and overlapping processes for the Triangular System Solver (TS) on a heterogeneous system consisting of a Huawei Kunpeng [8] ARM multi-core CPU and an Ascend 910 [3] AI HW accelerator. We propose an LLVM compiler tool-chain that a) leverages compiler analysis and b) utilizes novel performance models exploring recursive, iterative, and blocked computation models. Our tool-chain facilitates a speedup of up to 16x compared to an optimized 48-core CPU-only implementation.**

***Index Terms*—heterogeneous systems, accelerators, compiler, automation, DSE, cloud computing**

I. Introduction

Heterogeneous computing aims to exploit the strengths of diverse processor types and architectures to achieve superior performance, power efficiency, and cost-effectiveness than a homogeneous, general purpose CPU-based system. The term Supercloud [4] has been used to describe the next generation of the cloud architectures where demanding processing of AI, big data and HPC applications can be supported. In a large-scale heterogeneous computing environment, such as a Supercloud, high-speed interconnects enable different processors to collaborate efficiently using software to distribute and coordinate workloads at various levels of granularity, ranging from low-level hardware acceleration to high-level task scheduling and load balancing.

However, designing complex Supercloud heterogeneous systems poses significant challenges that are both time-consuming and demanding. Engineers must carefully consider physical resource constraints, communication costs, and other factors when deciding which portions of an application should be accelerated on GPUs or hardware accelerators (including TPUs, DPUs, etc.) and which should run on general purpose CPUs. Moreover, applications in many relevant domains, e.g., Deep Learning, Extended Reality (XR) or Autonomous Vehicles, offer opportunities for various forms of parallel execution, including Instruction Level (ILP), Task Level (TLP) and Pipeline (PP) Parallelism.

These different forms of parallelism, if harnessed for hardware acceleration, can result in significant speedups. Thus, a Design Space Exploration (DSE) methodology based on performance estimation models can be crucial. It can (a) utilize all the forms of parallelism mentioned above, (b) be driven directly by the application source code or a graph representation of a computation model (e.g., recursive, iterative, and blocked), (c) automatically determine the parts of the application that should be accelerated in hardware, and (d) carry out performance estimation.

II. Related Work

Monil *et al.* [5] introduce LaRIS, a portable framework for LAPACK functionalities on Heterogeneous Systems. LaRIS uses IRIS [5] to dynamically select the vendor-library kernel and suitable processor architecture at run-time, making the orchestration simpler when different architectures coexist in a single node. Valero-Lara [7] evaluates the use of OpenMP tasking with target GPU offloading as a potential solution for programming productivity and performance on heterogeneous systems. As a test case, the Triangular Solver (TS) routine is used. In both the aforementioned works, there is not a model to determine the refinement level and the expected performance.

In [6], the authors introduce ReLAPACK, a methodology that makes use of recursive algorithms in order to implement LAPACK functionalities. They claim that, contrary to blocked algorithms, recursive algorithms do not require significant tuning to define the proper granularity/refinement level. Conversely to our work, they target shared memory architectures and not heterogeneous systems.

III. Compiler Tool-chain

To address all the challenges mentioned in the previous section, we present **ReDSEa** (Recursive Design Space Exploration Automation): A tool-chain based on the LLVM [2] Compiler Infrastructure (Version 14.0.0).

A. Compiler Analysis

The first stage of our methodology is analyzing the application that is going to be mapped to a heterogeneous system or cloud architecture. The analysis phase includes an automated process that provides our compiler infrastructure with necessary information so as to guide the subsequent steps (Cost Models and Design Space Exploration) of the automation methodology. It can be viewed as the **input** to our automated system. The analysis is performed by generating the respective LLVM-IR from the application source code (typically written in C, C++). A number of LLVM analysis passes, developed within the scope of ReDSEa, analyze the intermediate representation of the applications and estimate the latency due to computation of every potential task, or otherwise described, of every node of the data flow graph. Furthermore, the communication cost is estimated by extracting the amount of data that is read and stored by every task. The data requirements, along with the available bandwidth of a target architecture, allows for an estimation of the latency, due to communication.

B. Cost Models

To obtain an early evaluation of the potential performance of every computational node, we need to introduce evaluation/cost models that can perform estimations of both computation and communication latency for the components of a Supercloud heterogeneous system, such as CPUs, GPUs, HW accelerators, etc.

To estimate the latency of a computational node that is mapped to multiple CPUs, GPUs and/or HW accelerators, we need to estimate 1) the latency of the computation on the CPUs, 2) the critical path of the computation that is offloaded to GPUs and/or HW accelerators, 3) the latency to transfer the data from the host main memory to the HW accelerator (Host-to-Device, H2D) and vice versa (Device-to-Host, D2H), and 4) the synchronization/invocation overhead. $Latency = CPUComputation + HWComputation + Communication + Synchronization/Invocation$

Let $S = \{S_1, S_2, \ldots, S_N\}$ be a set of nodes (tasks), with associated HW computation latency ($HWcomp_i$), HW communication latency ($HWcomm_i$) and synchronization/invocation overhead ($OVHD_i$). For every node i the cumulative latency will be $HW_i = HWcomp_i + HWcomm_i + Synch_i \mid i = 1, \ldots, N$.

C. Design Space Exploration

Based on the cost models of the previous subsection, a list of candidates for acceleration is generated. The selection (branch-and-bound) algorithm recursively explores the subsets of the list of candidates, in a similar manner to the Bron-Kerbosch algorithm [1]. The output returned is the set with the highest speedup (minimum cumulative latency) that stays within the user defined resource budget, which is translated as the amount of resources available for hardware acceleration.

IV. Triangular System Solver

Solving Triangular Systems is a fundamental problem from the dense linear algebra domain. In this example, we solve the linear system $Lx = b$, where L is a dense lower-triangular $n \times n$ matrix and b is a dense vector of length n.

The solution to this problem, in order to explore multiple levels of granularity, relies on dividing b into an upper and a lower half and the matrix L into matrices $L.up$, $L.mid$, and $L.low$ corresponding to the upper left, lower left, and lower right blocks of L, as seen in Figure 1. $L.up$, $L.low$ are dense

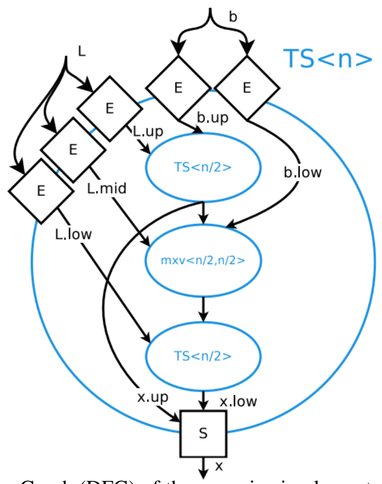

Fig. 1. Data Flow Graph (DFG) of the recursive implementation of triangular system solver `TS<n>` and its refined (decomposed) nodes `TS<n/2>` and the matrix-vector multiplication update `mxv<n/2,n/2>`.

lower-triangular $n/2 \times n/2$ matrices and $L.mid$ is a full dense $n/2 \times n/2$ matrix. With this analysis, the system can be solved in 3 stages:

1) Solve the triangular system $(L.up)(x.up) = b.up$
2) Update $b.low = b.low - (L.mid)(x.up)$
3) Solve the triangular system $(L.right)(x.low) = b.low$

The solution to the original system is $x = (x.up, x.low)$.

We extend the problem by solving n linear systems for n different b vectors of size n while keeping the same L lower-triangular $(n \times n)$ matrix.

V. Models of Computation

Three models of computation are explored within the scope of this work: Recursive, as seen in [6], Iterative and Blocked.

A. Recursive

The recursive model of computation offers a decomposition of the initial problem to several tasks, as shown in Figure 1. It also provides the opportunity to expose parallelism and finally to explore various levels of granularity in order to make the best architectural decisions and use efficiently any given available software (general purpose CPUs) and hardware accelerators resources.

In the example of the Triangular System (TS), the initial problem of size n is decomposed to the tasks: `TS<n/2>`, general matrix-vector multiplication `gemv<n/2,n/2>` and `TS<n/2>`, along with the respective dependencies. The gemv task offers data level parallelism and, hence, can be an excellent candidate to be accelerated. As we solve n instances of TS for n vectors of b, the gemv task becomes a matrix-matrix multiplication task and the second stage of the problem is transformed to a general matrix multiplication (gemm) task.

The refined `TS<n/2>` tasks can be refined as well, so as to expose more parallelism and accelerate a larger part of the initial `TS<n>`. For every iteration i of the decomposition/refinement of TS, the size of the initial matrix $n \times n$ $(i = 0)$ is decreased to a quarter of the previous iteration (e.g. $n/2 \times n/2$ for $i = 1$, $n/4 \times n/4$ for $i = 2$ etc.).

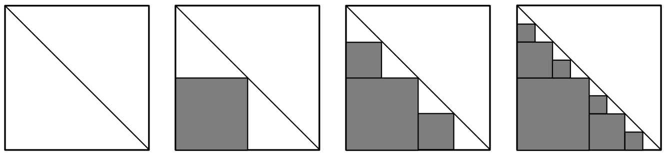

Fig. 2. Recursive model for Triangular System Solver (Iteration: 0, 1, 2 and 3 Refinement Level: 1, 2, 4 and 8). Each grey rectangle (square) represents the gemm computation that is offloaded to an accelerator. The white triangles represent the part of the computation (TS) that remains in the host CPU.

The next refinement stage is applied *only* if it benefits performance, i.e., if the following **condition** is satisfied: $2 \times TS(i+1) < TS(i) \mid i = 0, \ldots, N$
We define as $r(i)$ the **Refinement Level** of iteration i.

$$r(i) = 2^i \mid i = 0, \ldots, N \tag{1}$$

The models that estimates the performance, i.e., computation and communication latency, on a heterogeneous architecture are described by the following formulas:

$$Comp(i) = r(i) \times \ TS(i) + \sum_{j=0}^{i-1} r(j) \times \ gemm(j)$$
$$Comm(i) = \sum_{j=0}^{i-1} r(j) \times Comm_{(H2D+D2H)}(j)$$

B. Iterative

Once the granularity of the TS computation that remains on the host has been determined, an iterative model of computation could be offered as an alternative.
There are two main reasons to favor an iterative approach over a recursive one: a) It can offer better utilization of the accelerators that are available, as fewer accelerators are dedicated to compute relatively smaller parts of the computation compared to a recursive one which allocates smaller and smaller parts of the computation to be computed by the HW accelerators. b) It demands less engineering effort to be implemented while at the same time the performance is not sacrificed, but instead it is kept equal, or slightly better, compared to a recursive one.

The models that estimate the expected performance and communication cost of the iterative approach are:

$$Comp(i) = r(i) \times \ TS(i) + \sum_{j=0}^{r(i)-2} gemm(i,j)$$
$$Comm(i) = \sum_{j=0}^{r(i)-2} (Comm_{H2D}(j) + Comm_{D2H}(i))$$

C. Blocked

A blocked approach, as seen in Figure 4, can be used once the level of refinement and granularity of the TS computations that reside at the host CPU has been determined.

The advantages of a blocked model, compared to a recursive or an iterative one, are better overall load balancing, more efficient use of HW accelerators resources, and better scheduling.

The workflow of the blocked model (Figure 5) allows a computation in rounds offloading equivalent workloads for the acceleration of gemm in every round and using the available resources efficiently.

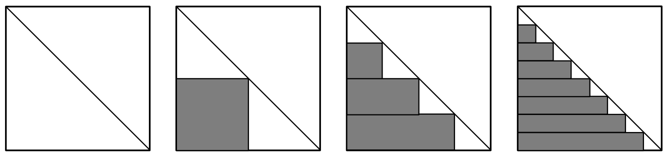

Fig. 3. Iterative model for Triangular System Solver (Iteration: 0, 1, 2 and 3 Refinement Level: 1, 2, 4 and 8). Each grey rectangle represents the gemm computation that is offloaded to an accelerator. The white triangles represent the part of the computation (TS) that remains in the host CPU.

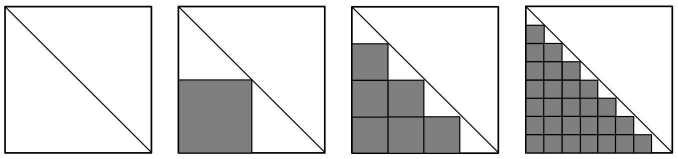

Fig. 4. Blocked model for Triangular System Solver (Iteration: 0, 1, 2 and 3 Refinement Level: 1, 2, 4 and 8). Each grey rectangle (square) represents the gemm computation that is offloaded to an accelerator. The white triangles represent the part of the computation (TS) that remains in the host CPU.

For every iteration i and a respective refinement level $r(i)$, the number of **rounds of computation** is $r(i) - 1$ and the **blocks of computation** is $\frac{r(i)}{2}$ per round, so at to have equal workloads per round. Also, the partitioning of the computation into blocks unlocks the potential for parallelism execution with multiple accelerators, requiring less engineering effort compared to the previous models, and it unlocks the option to overlap the acceleration of gemm with the CPU computation of TS.

Thus, the total number of blocks (gemm(i) computations) to be accelerated are $(r(i)-1) \times \frac{r(i)}{2}$ (rounds of computation multiplied by the blocks of computation per round). In the example of Figure 5, the total number of blocks is $7 \times \ 4 = 28$.

The models that compute the expected performance and communication cost of the blocked approach are:

$$Comp(i) = r(i) \times \ TS(i) + ((r(i)-1) \times \frac{r(i)}{2}) \times \ gemm(i)$$
$$Comm(i) = ((r(i) - 1) \times \frac{r(i)}{2}) \times \ Comm_{(H2D+D2H)}(i)$$

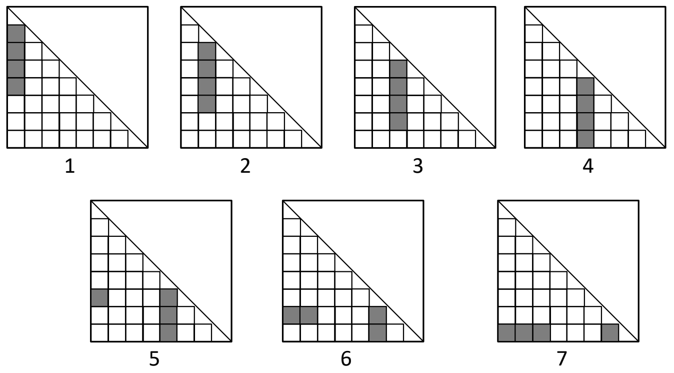

Fig. 5. Workflow in seven rounds of computation for the blocked model of Triangular Solver (Refinement: $r(3) = 8$, Rounds: $r(3) - 1 = 7$, Blocks: $\frac{r(3)}{2} = 4$).

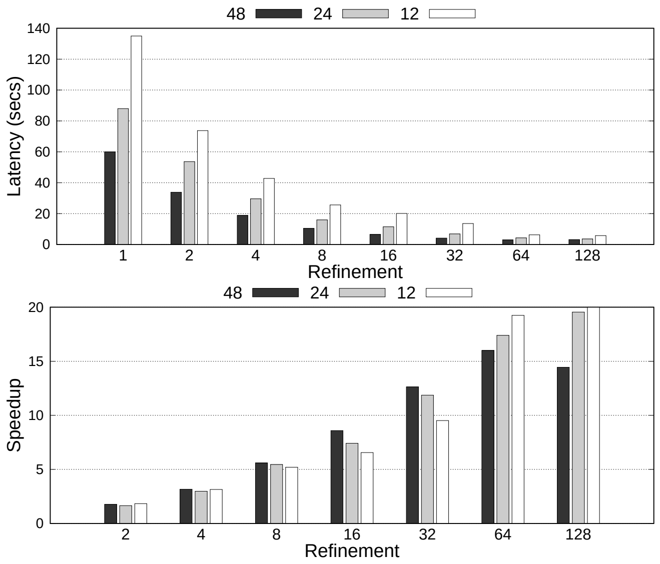

Fig. 6. Latency (top) and Speedup (bottom) obtained for Triangular System (TS) Solver increasing the refinement level and gradually offloading more of the initial computation to the Ascend device. The computation remaining in the host CPU was computed using 48, 24 and 12 cores respectively.

VI. Experimental Results

Our experimental setup consists of a general purpose 48-core ARM CPU (Huawei Kunpeng [8]) and a Huawei Ascend 910 AI processor [3], that consists of 32 Da Vinci AI cores with a peak performance of 320 TFLOPS. The two devices communicate via a PCIe bus. The results are equivalent for all three computation models explored. In Figure 6 (top) the total latency of every heterogeneous design can be seen while the refinement level increases and while using either all available CPU resources (48 cores), half of them (24 cores) or finally 12 of the available CPU cores. By increasing the refinement level, hence offloading a larger part of the computation to Ascend, significant time can be saved even when using fewer CPU cores, e.g., with refinement equal to 32 and 12 CPU cores.

The respective speedup over an optimized CPU-only baseline, showcased in Figure 6 (bottom) can be up to a compelling **16x** using 48 CPU cores (refinement=64). However, the speedup decreases with the next iteration of refinement (128), which also employs finer granularity. This is due to two main factors: a) The CPU-residing part of the TS computation is so fine-grained that it cannot be executed faster than the previous iteration. So, the condition $2 \times TS(i+1) < TS(i) \mid i = 0, \ldots, N$, as defined in Section V-A is not satisfied and the refining process ends. This can be observed in Figure 7, where the CPU latency (48 cores) of the last refinement iteration (128) is larger than the previous iteration. b) The communication cost raises substantially while refinement increases. The communication latency between the CPU host and the Ascend device at the last two refinement iterations (64 and 128) surpasses the cost of the CPU computation resulting in significant overhead and halts the potential for more speedup.

VII. Conclusions

Using the ReDSEa tool-chain, which incorporates performance models derived from recursive, iterative, and blocked computation models, we have managed to automatically map the Triangular Solver (TS) onto a heterogeneous system, consisting of a Kunpeng 48-core ARM CPU and an Ascend AI accelerator device, achieving up to a 16x speedup. We have explored three models of computation with an emphasis on the blocked approach to minimize HW acceleration times compared to the recursive and iterative versions.

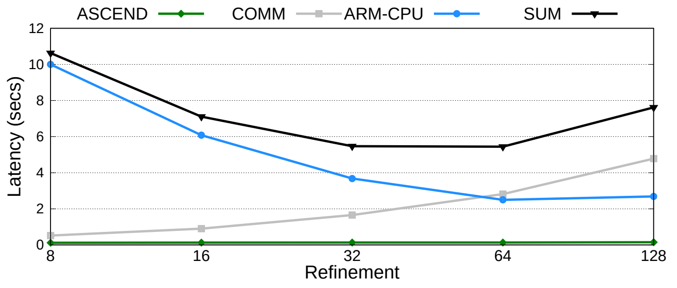

Fig. 7. Latency of Ascend accelerator computation, host-to-device and device-to-host communication, ARM CPU (48 cores) computation and their sum.

VIII. Future Directions

Our objective is to investigate the potential of parallelism and overlapping in the blocked model, aiming to showcase their impact on the overall latency and speedup achieved by the respective designs. We also plan to extend this methodology to other applications in the HPC and AI domains, such as Dense Cholesky Factorization, QR Matrix Factorization, and others. Developing new models or extending existing ones will be necessary to estimate the performance of more applications. Finally, we aim to apply this methodology to more complex heterogeneous and Supercloud architectures.

References

[1] Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. In *Communications ACM*, volume 9, pages 575–577, 1973.

[2] Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the 2nd International Symposium on Code Generation and Optimization*, pages 75–88, March 2004.

[3] Heng Liao, Jiajin Tu, Jing Xia, Hu Liu, Xiping Zhou, Honghui Yuan, and Yuxing Hu. Ascend: a scalable and unified architecture for ubiquitous deep neural network computing : Industry track paper. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 789–801, 2021.

[4] Bill McColl. Superclouds: Scalable high performance nonstop infrastructure for ai and smart societies. In *International Conference on Smart Cities, Infrastructure, Technologies and Applications*, pages 3–5. Springer, 2017.

[5] Mohammad Alaul Haque Monil, Narasinga Rao Miniskar, Frank Y. Liu, Jeffrey S. Vetter, and Pedro Valero-Lara. Laris: Targeting portability and productivity for lapack codes on extreme heterogeneous systems by using iris. In *2022 IEEE/ACM Redefining Scalability for Diversely Heterogeneous Architectures Workshop (RSDHA)*, pages 12–21, 2022.

[6] Elmar Peise and Paolo Bientinesi. Algorithm 979: Recursive algorithms for dense linear algebra—the relapack collection. *ACM Trans. Math. Softw.*, 44(2), sep 2017.

[7] Pedro Valero-Lara, Jungwon Kim, Oscar Hernandez, and Jeffrey Vetter. Openmp target task: Tasking and target offloading on heterogeneous systems. In *Euro-Par 2021: Parallel Processing Workshops*, pages 445–455, 2022.

[8] Jing Xia, Chuanning Cheng, Xiping Zhou, Yuxing Hu, and Peter Chun. Kunpeng 920: The first 7-nm chiplet-based 64-core arm soc for cloud services. *IEEE Micro*, 41(5):67–75, 2021.