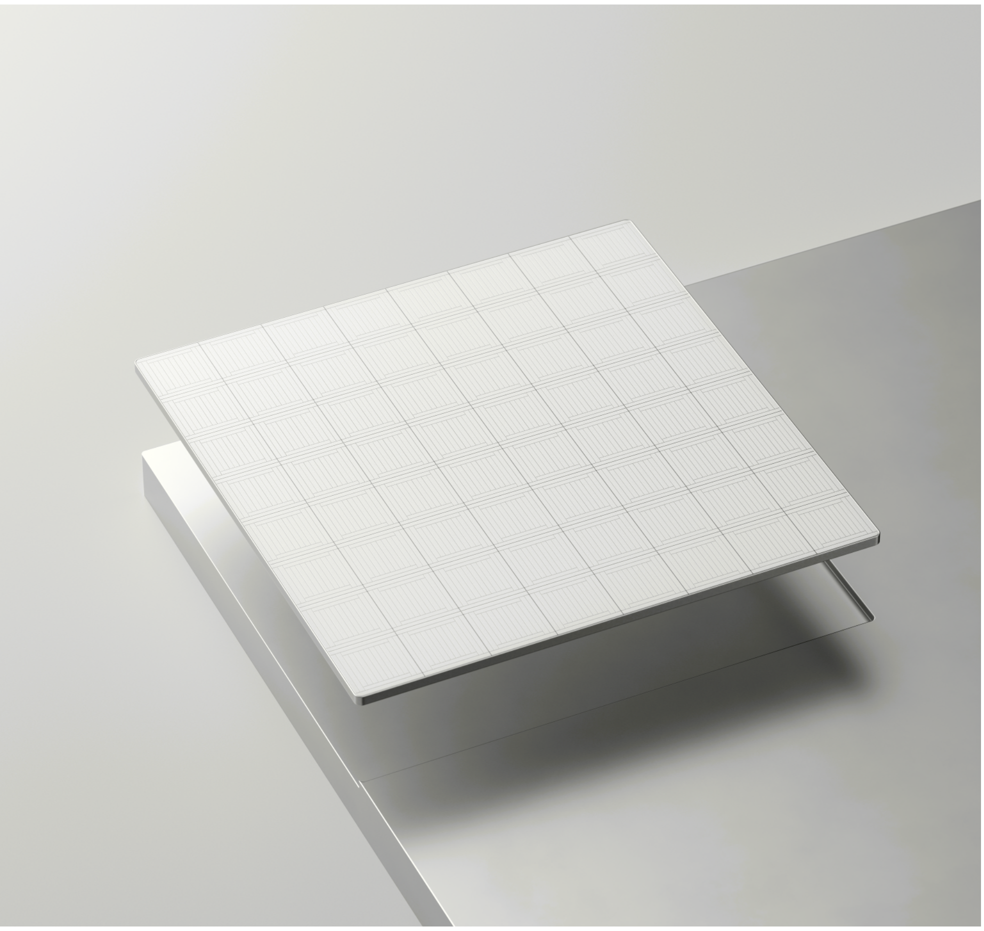
Design goals, advantages and benefits of Flow Computing

Executive Summary

Flow Computing is a revolutionary technology for boosting the performance of future processors and increasing the productivity of parallel software engineering over current processors. It can be applied widely to current processor technology as a parallel computing accelerator for CPUs or as a fully integrated next generation CPU.

We provide this document to briefly summarize the design goals, organization, hardware advantages and methodological benefits of Flow Computing technology.



Martti Forsell
CTO, Chief Architect
Flow-computing

Table of Contents

Summary of Flow Computing design goals and HW/SW advantages

Flow Computing is a revolutionary technology for boosting the performance of future processors and increasing the productivity of parallel software engineering over current processors. It can be applied widely to current processor technology as a parallel computing accelerator for CPUs or as a fully integrated next generation CPU.

There are no detailed architecture publication of our patented Flow Computing technology integrating design principles, architectural and methodological aspects to a coherent presentation with respect to current CPUs. We provide this document to briefly summarize the design goals, hardware advantages and methodological benefits of Flow Computing technology.

Main design goals

The main design goals of Flow Computing include (i) high performance in general purpose parallel computing, (ii) increased software engineering productivity for parallel functionalities while maintaining full backwards compatibility with the existing software base and tools as well as (iii) flexibility /scalability that allows it to be applied widely to different use cases, instruction sets, processor manufacturers and ICT devices/appliances.

Experimental Flow Computing systems have been implemented with a parametric clock cycle and RTL-accurate software simulator TPASim and a hardware instance of a 32-bit 6-FU 16-BE proof-of-concept has been realized on FPGA. According to our tests, the FPGA is giving identical results to the software simulator.

We have made early performance, programmability and scalability tests with selected Flow processor configurations [Forsell22, Forsell23a]. The results indicate that the goals (i) and (ii) as well as partially also (iii) appear to be achievable including our main value promise of being able to provide 100x performance boost over current systems.

Organization of a Flow system

A Flow superCPU consists of a CPU from our processor partner and an add-on Parallel Processing Unit (PPU) that is attached to the CPU. To emphasize the different roles of superCPU components in fetching instructions and their main usage, CPU is called as Frontend or Sequential Processing Unit (SPU) and PPU is called Backend. Both SPU and PPU will have internal cache memory. A Flow system consists of a superCPU and memory system (see Figure 1). Since the PPU speeds up the system by a large margin, most system designers want to increase the memory bandwidth.

Hardware advantages

To highlight the hardware advantages of Flow over traditional SMP/NUMA CPU (and GPU) computing, let us have a more detailed look at a number of differences between them (with 10 first illustrated in Figure 2):

1. Nonexistent cache coherence issues. Unlike in current CPU systems, in Flow's architecture there are no cache coherence issues in the memory systems due to the memory organization excluding caches in the front of the intercommunication network [Forsell16a].

2. Cost efficient synchronization. Flow synchronization cost is roughly 1/Tb (Tb=fibers per PPU core) [Forsell23a] whereas in SMP/NUMA CPU systems it can be hundreds to thousands of clock cycles [Forsell22, Forsell23a] and in GPUs it can be from thousands to hundreds of thousands of clock cycles [Zhang20].

3. Support for parallel computing primitives. Flow's architecture provides unique and specific techniques/solutions for executing concurrent memory access (both read and write) operations [Forsell18a], multi-operations for executing reductions [Forsell23b], multi-prefix operations [Forsell23b], compute-update operations [Forsell20] and fiber mapping operations [Forsell18b] in the most efficient manner possible. These primitives are not available in current CPUs. Implementing these primitives in Flow Computing involves active memory technologies in the SRAM-based on-chip shared cache level providing potentially better performance and greater flexibility than current DRAM-based processing in memory (PIM) solutions, but not preventing the use of both techniques in the same design.

4. Flexible threading/fibering scheme. Flow Computing technology allows an unbounded number of fibers at the model level, which can also be supported in hardware (within certain bandwidth constraints). In current-generation CPUs, the number of

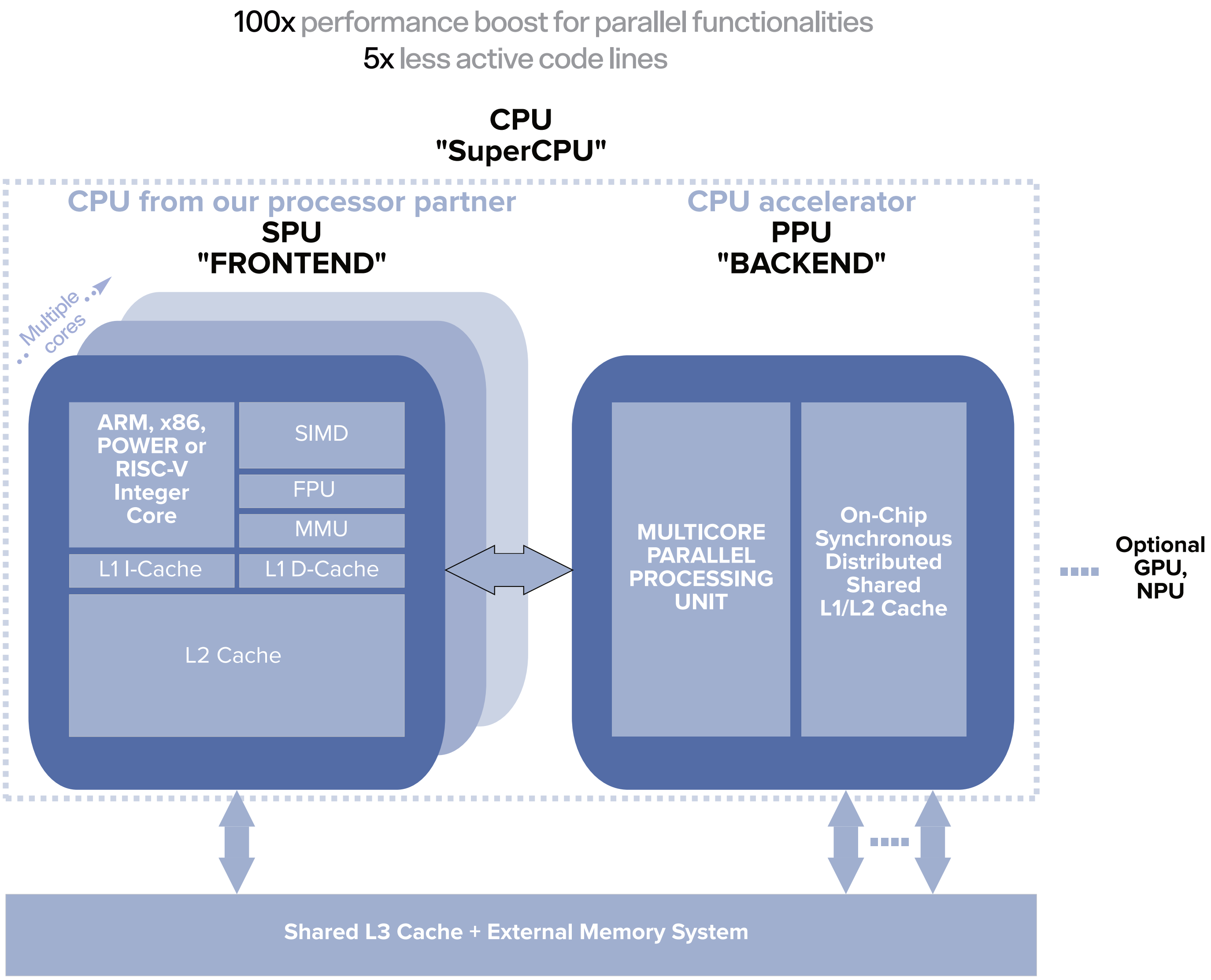

100x performance boost for parallel functionalities
5x less active code lines
CPU
"SuperCPU"
CPU from our processor partner
SPU
"FRONTEND"
CPU accelerator
PPU
"BACKEND"
Multiple cores
ARM, x86, POWER or RISC-V Integer Core
SIMD
FPU
MMU
L1 I-Cache
L1 D-Cache
L2 Cache
MULTICORE PARALLEL PROCESSING UNIT
On-Chip Synchronous Distributed Shared L1/L2 Cache
Optional GPU, NPU
Shared L3 Cache + External Memory System

Figure 1. Flow superCPU.

threads is - in theory - not bounded, but if the number of hardware threads is exceeded in the case of interdependencies, the results can be very bad [Forsell22, Forsell23a]. In addition, the operating systems typically limit the number of threads to a few thousand at most. The mapping of fibers to backend units is a programmable function allowing further performance improvements in Flow [Forsell18b].

5. Low-level parallelism for dependent operations. In Flow-enabled CPUs, it is possible to execute dependent operations with the full utilization within a step (with the help of chaining of functional units), whereas in current CPUs the operations executed in parallel need to be independent due to parallel organization of the functional units [Forsell02].

6. Non-existent context switching cost. In Flow-enabled CPUs, the fiber switching cost is zero whereas in current CPUs it is more than 100 clock cycles [DEDICAT23].
7. Intercommunication traffic congestion avoidance. In Flow-enabled CPUs, the probability of inter-communication network traffic congestion is low due to hardware support for hashing, concurrent memory access, multi-operations and multi-prefix operations. In current CPUs, congestion can happen frequently if the access patterns are non-trivial [Forsell23a].

8. Scalable latency tolerance. Flow’s technology provides a scalable latency hiding mechanism for constellations up to thousands of backend units, whereas in current CPUs the latency tolerance (with snooping/directory-based cache coherence maintenance mechanisms) appears to be relatively weakly scalable [Forsell23a]. There is also evidence that in high-end Flow-enabled systems, even the latency of DRAM -based memory systems can be hidden in many cases with suitable memory organization and sufficient bandwidth [Forsell10].

9. No need for locality-maximizing memory data partitioning. Flow-enabled CPUs are

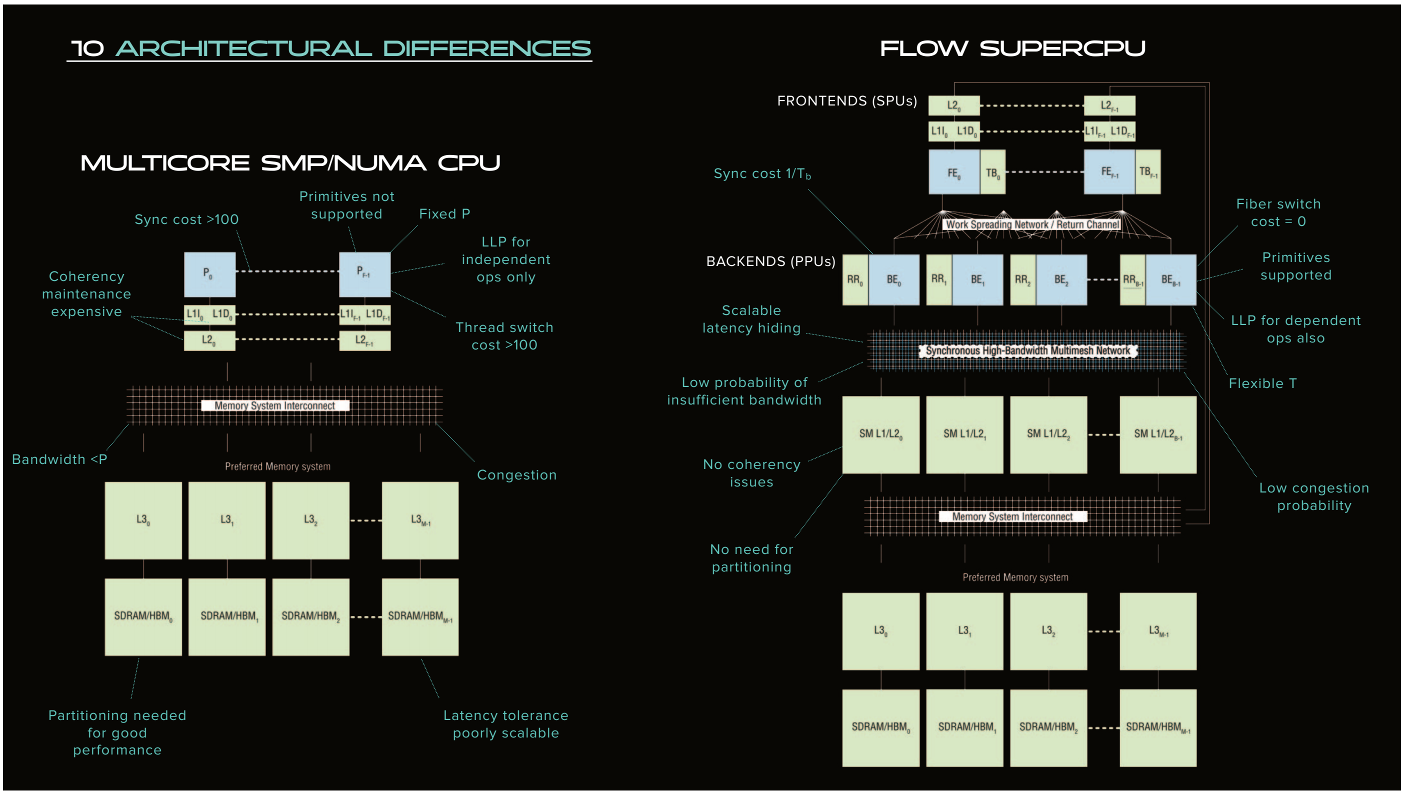


Figure 2. Architectural differences current multicore SMP/NUMA CPUs and Flow superCPU systems.

not prone to alternative memory partitioning schemes, such as interleaving, hashing, stacking and blocking, whereas current CPUs are highly-sensitive to data placement [Forsell23a].

10. Sufficient intercommunication bandwidth. Flow's PPU has an intercommunication network that is designed to provide sufficient bandwidth for random communication, whereas current CPUs are limited only to cases where most references are local [Forsell23a]. This kind of locality maximization is not always possible since in the general case there is no algorithm to maximize locality.

11. Dual unit organization. Current multicore CPUs were built by replicating processors originally designed for sequential computing and therefore optimized for low latency [Bloch59, Tomasulo67, Forsell96]. As a result, they are relatively good for executing sequential workloads but have substantial performance issues with non-trivial parallel functionalities.

To support high-speed execution of parallel code, Flow introduces the PPU that utilizes the slack of parallelism to reorganize operations so that the requirement for low-latency is turned to the need for high throughput [Forsell02, Forsell16b] and combines it with a CPU.

The resulting dual unit CPU-PPU combines the best of both worlds to achieve the best performance for modern workloads containing a lot of parallelism but also some sequential parts while providing backwards compatibility via the CPU.

12. Minimal disadvantages of superpipelining while having all benefits of that and full support for long latency, floating point and application-specific operations. Flow Computing PPU is a fully superpipelined design with regular structure and patented support for long-latency operations, floating point operations and optional application-specific operations [Forsell02, Forsell16b, Forsell18c, Forsell19]. In current CPUs, superpipelining increases the degree of pipeline delays that may cannibalize the performance benefits.

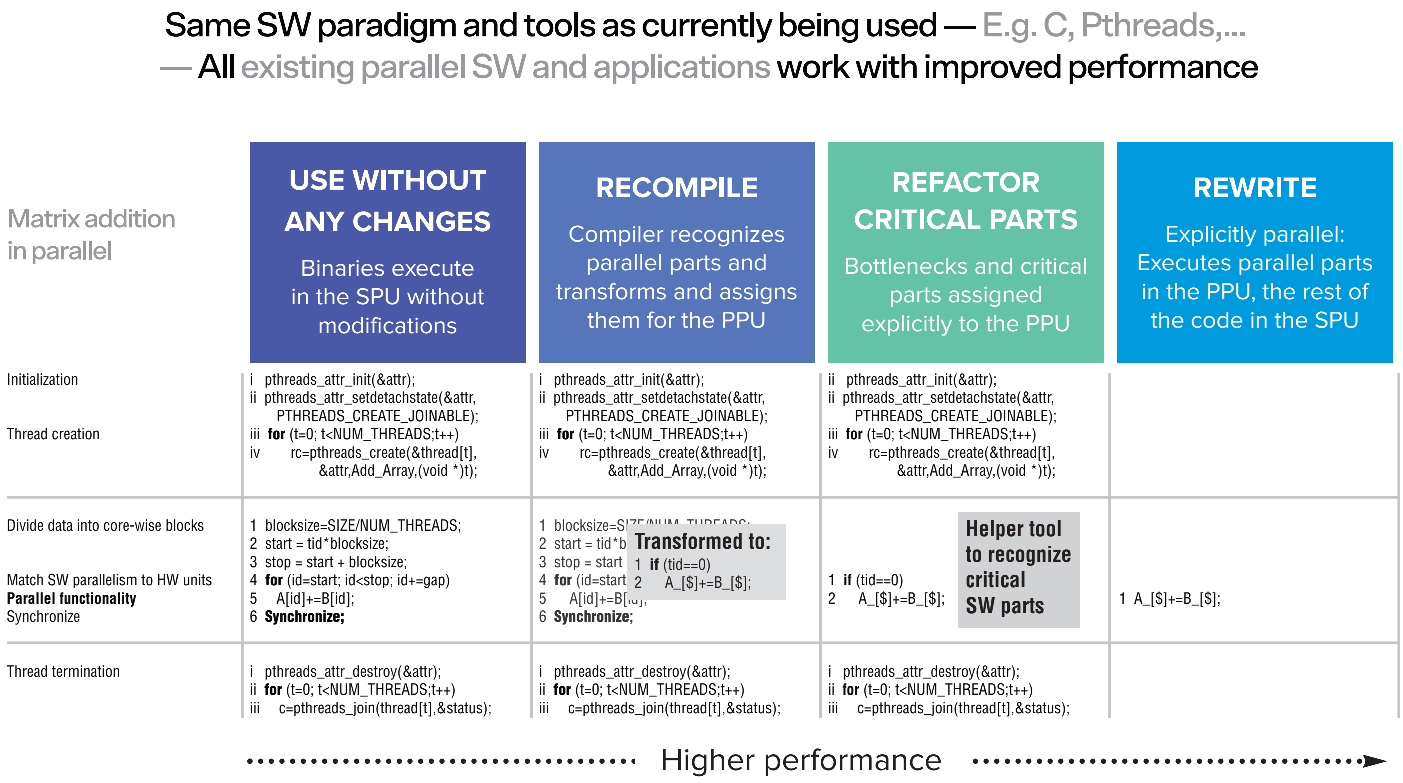
13. Parametric design and instruction set independency. Flow is not limited to single instances only, as it features parametric blocks with design time adjustable numbers of CPU cores, numbers of PPU cores, numbers and types of functional units per PPU core, size and organization of step caches, scratchpads, on-chip shared caches, latency compensation unit length, instruction set etc. Current CPU designs are typically tied to certain instruction sets and may require partial redesign if these kinds of parameters are altered.

14. Support for the key patterns of parallel computation. Flow supports the key patterns of parallel computation, such as parallel execution, reduction, spreading and permutation [Forsell23c]. Current-generation CPUs support only the parallel execution pattern.

Methodological benefits

At the methodological side, the main benefits include:

A. Backwards compatibility with the existing software base. The Flow technology uses a dual unit organization dividing the hardware to relatively tightly-coupled frontend (FE) and backend (BE) units. The former are aimed for executing sequential parts of the code (control of TCFs, base ad-



Same SW paradigm and tools as currently being used — E.g. C, Pthreads,...
— All existing parallel SW and applications work with improved performance

Matrix addition in parallel	USE WITHOUT ANY CHANGES Binaries execute in the SPU without modifications	RECOMPILE Compiler recognizes parallel parts and transforms and assigns them for the PPU	REFACTOR CRITICAL PARTS Bottlenecks and critical parts assigned explicitly to the PPU	REWRITE Explicitly parallel: Executes parallel parts in the PPU, the rest of the code in the SPU

Initialization
Thread creation
	pthreads_attr_init(&attr);
	pthreads_attr_setdetachstate(&attr, PTHREADS_CREATE_JOINABLE);
	for (t=0; t<NUM_THREADS;t++)
	rc=pthreads_create(&thread[t], &attr,Add_Array,(void *)t);
		pthreads_attr_init(&attr);
		pthreads_attr_setdetachstate(&attr, PTHREADS_CREATE_JOINABLE);
		for (t=0; t<NUM_THREADS;t++)
		rc=pthreads_create(&thread[t], &attr,Add_Array,(void *)t);
			pthreads_attr_init(&attr);
			pthreads_attr_setdetachstate(&attr, PTHREADS_CREATE_JOINABLE);
			for (t=0; t<NUM_THREADS;t++)
			rc=pthreads_create(&thread[t], &attr,Add_Array,(void *)t);	

Divide data into core-wise blocks
Match SW parallelism to HW units
Parallel functionality
Synchronize
	blocksize=SIZE/NUM_THREADS;
	start = tid*blocksize;
	stop = start + blocksize;
	for (id=start; id<stop; id+=gap)
	A[id]+=B[id];
	Synchronize;
		blocksize=SIZE/NUM_THREADS;
		start = tid*b
		stop = start
		for (id=start
		A[id]+=B[id];
		Synchronize;
		Transformed to:
		if (tid==0)
		A_[$]+=B_[$];
			if (tid==0)
			A_[$]+=B_[$];
			Helper tool to recognize critical SW parts
				A_[$]+=B_[$];

Thread termination	pthreads_attr_destroy(&attr);
	for (t=0; t<NUM_THREADS;t++)
	c=pthreads_join(thread[t],&status);
		pthreads_attr_destroy(&attr);
		for (t=0; t<NUM_THREADS;t++)
		c=pthreads_join(thread[t],&status);
			pthreads_attr_destroy(&attr);
			for (t=0; t<NUM_THREADS;t++)
			c=pthreads_join(thread[t],&status);	

Figure 3. Stepwise migration from current multicore software base to full exploitation of Flow Computing.

dress computation) and parts that do not have enough parallelism for efficient execution in the backend. The frontend designs are provided by our processor partners and therefore provide full backwards compatibility with the existing software base and tools. The PPU processing elements execute the individual parallel fibers of the code. Many new architectural innovations are not able to provide backwards compatibility.

B. Well-founded theory of parallel algorithms. Flow supports the Parallel Random Access Machine -style model of computation [Fortune78, Forsell13] featuring a well-founded and implementation-independent theory of parallel algorithms [Jaja92, Keller01]. This is not directly applicable to current CPUs or GPUs which require much more architecture and configuration dependent theories. The TCF model, merging together homogeneous threads flowing through the same control path [Leppänen11], simplifies further programming of parallel functionalities, eliminates unnecessary replication of hardware and data, as well as serves as a central model from which many of the existing models and be deduced [Forsell13].

C. Well-defined state of computation. In Flow systems, the state of computation is well-defined due to synchrony of executed fibers and strict memory consistency whereas in current CPUs and GPUs execution of threads is intrinsically asynchronous and memories are weaker implementing relaxed consistency. The well defined state simplifies programming, makes it less error-prone and helps in verification of correctness.

D. Greatly simplified programming of parallel functionalities. Usage of Flow Computing technology greatly simplifies programming compared to current CPUs and GPUs since unlike in them (i) most explicit synchronizations can be eliminated due to inherently synchronous execution of fibers, (ii) in many cases, looping is not needed to match the software to the available hardware threads, (iii) in many cases there is no need for maximizing the locality of data references, and (iv) there is no need to create/terminate fibers while the degree of parallelism can be adjusted dynamically according to needs of the executed algorithm [Forsell22, Forsell23a].

E. Plans for a step-wise migration path. We plan to provide a four-step migration path from popular parallel computing methodologies to full-fledged Flow Computing (see Figure 3): (i) Flow systems will provide full binary-level backwards compatibility with the existing software base and tools with current performance level via the frontend. (ii) If the current programs are recompiled for the Flow system with our compiler (currently under development), the compiler recognizes patterns that can easily be targeted for backend execution and compiles the code accordingly leading to increased performance. We plan to port a set of key libraries to utilize Flow Computing so that if the programs employs them, it will have further performance boost. (iii) We aim to provide a tool for helping migration by recognizing additional patterns that can be potentially executed in the backend with a help of the programmer. (iv) Full performance and simplicity of native and natural parallel programming can be achieved if the application is written for the Flow system from the beginning. This simplifies greatly the parallel parts of the program. Utilizing this for future software development (and high school/university education) improves the productivity of software engineering and makes usage of explicit parallel algorithms available also for average programmers.

F. Possibility for special optimizations. If required for performance or efficiency reasons, it is also possible to switch off the hashing of memory addresses, step-wise synchronization and bound the thickness of TCFs for traditional locality-optimized NUMA computation. The hashing is designed so that hashed and non-hashed memory regions can co-exist in the same memory space.

REFERENCES

[Bloch59] E. Bloch, The engineering design of the Stretch computer, Proc. of the Fall Joint Computer Conference, 1959, 48-59.
[DEDICAT23] A. Anttonen, Y. Carlinet, P. Demestichas, M. Forsell, V. Lamprousi, G. Lecker Ricardo, K. Mößner, N. Perrot and V. Stavroulaki, Reinout Eyckerman, Phil Reiter and J. Yang, Deliverable D3.3 "Final release and lab test report of mechanisms for dynamic distribution of intelligence", Dynamic coverage Extension and Distributed Intelligence for human Centric Applications with assured security, privacy and Trust: from 5G to 6G (DEDICAT 6G), 2023. To appear in https://dedicat6g.eu/results/deliverables/.
[Forsell96] M. Forsell, Minimal Pipeline Architecture-an Alternative to Superscalar Architecture, Microprocessors and Microsystems 20, 5 (1996), 277-284.
[Forsell02] M. Forsell, Architectural differences of efficient sequential and parallel computers, Journal of Systems Architecture 47, 13 (July 2002), 1017-1041.
[Forsell10] M. Forsell, Performance comparison of some shared memory organizations for 2D mesh-like NOCs, Microprocessors and Microsystems 35, 2 (March 2011), 274-284.
[Forsell13] M. Forsell and V. Leppänen, An Extended PRAM-NUMA Model of Computation for TCF Programming, Int. Journal of Networking and Computing 3, 1 (2013), 98-115.
[Forsell16a] M. Forsell, J. Roivainen and V. Leppänen, The REPLICA on-chip network, In the Proceeding of the 2016 IEEE Nordic Circouts and Systems Conference (NORCAS'16), November 1-2, 2016, Copenhagen, Denmark.
[Forsell16b] M. Forsell, J. Roivainen and V. Leppänen, Outline of a Thick Control Flow Architecture, In the Proceedings of the 5th Workshop on Parallel Programming Models Special Edition on Task Parallelism, October 26-28, 2016, Marina del Rey Marriott, Los Angeles, USA.
[Forsell18a] M. Forsell, J. Roivainen, V. Leppänen and J. Träff, Supporting Concurrent Memory Access in TCF Processor Architectures, Microprocessors and Microsystems 63, November 2018, 226-236.
[Forsell18b] M. Forsell, Flexible Fibering Scheme for Thick Control Flow Processors, In the Proceedings of the 24th Int'l Conf on Parallel and Distributed Processing Techniques and Applications (PDPTA'18), July 30-August 2, 2018, Las Vegas, USA.
[Forsell18c] M. Forsell, Architecture for Long Latency Operations in Emulated Shared Memory Architectures, US Patent 10127048B2, November 13, 2018.
[Forsell19] M. Forsell, Floating-point supportive pipeline for emulated shared memory architectures, European patent EP2866138B1 (Germany, France, UK, Italy), August 7, 2019
[Forsell20] M. Forsell, J Roivainen and J. Träff, Optimizing Memory Access in TCF Processors with Compute-Update Operations, In the Proceedings of 22nd Workshop on Advances in Parallel and Distributed Computational Models (APDCM'20) in conjunction with the 33rd IEEE International Parallel and Distributed Processing Symposium (IPDPS'20), May 18 – 22, 2020, New Orleans, Louisiana, USA.
[Forsell22] M. Forsell, S. Nikula, J. Roivainen, V. Leppänen and J. L. Träff, Performance and Programmability Comparison of the Thick Control Flow Architecture and Current Multicore Processors, Journal of Supercomputing 78, 3 (2022), 3152-3183. https://doi.org/10.1007/s11227-021-03985-0
[Forsell23a] M. Forsell, J. Roivainen, V. Leppänen and J. L. Träff, Preliminary Performance and Memory Access Scalability Study of Thick Control Flow Processors, In the Proceedings of 2023 IEEE Nordic Circuits and Systems Conference (IEEE NORCAS'23), October 31 - November 1, 2023, Aalborg, Denmark.
[Forsell23b] M. Forsell, J. Roivainen, V. Leppänen and J. L. Träff, Realizing Multioperations and Multiprefixes in Thick Control Flow Processors, Microprocessors and Microsystems 98, April 2023. https://doi.org/10.1016/j.micpro.2023.104807
[Forsell23c] M. Forsell, Overview of the Flow-computing technology, White Paper, VTT, April 2023.
[Fortune78] S. Fortune and J. Wyllie, Parallelism in Random Access Machines, Proc. 10th ACM STOC, Association for Computing Machinery, New York, 1978, 114-118.
[Jaja92] J. Jaja, Introduction to Parallel Algorithms, Addison-Wesley, Reading, 1992.
[Keller01] J. Keller, C. Keßler, and J. Träff, Practical PRAM Programming, Wiley, New York, 2001.
[Leppänen11] V. Leppänen, M. Forsell and J-M. Mäkelä, Thick Control Flows: Introduction and Prospects, Proc. 2011 Int. Conf. on Parallel and Distributed Processing Techniques and Applications, July 18-21, 2011, Las Vegas, USA, 540-546.
[Tomasulo67] R. Tomasulo, An efficient algorithm for exploiting multiple arithmetic units, IBM Journal of Research and Development 11, 1 (1967), 25-33.
[Zhang20] L. Zhang, M. Wahib, H. Zhang and S. Matsuoka, A Study of Single and Multi-device Synchronization Methods in Nvidia GPUs, Proc. IPDPS'20, May 18-22, 2020, New Orleans, Louisiana, USA.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH FLOW-COMPUTING LTD'S RESEARCH AND DEVELOPMENT RESULTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN FLOW-COMPTING LTD'S TERMS AND CONDITIONS OF SALE/TECHNOLOGY TRANSFER FOR SUCH RESULTS, FLOW-COMPTING LTD ASSUME NO LIABILITY WHATSOEVER AND FLOW-COMPTING LTD DISCLAIM ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF FLOW-COMPTING LTD'S RESEARCH AND DEVELOPMENT RESULTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Flow-computing Ltd. may make changes to specifications, architecture, and methodology descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undefined”. Flow-computing Ltd reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current version of the document and possible characterized errata are available on request.

May 29, 2024 (version 0.8 2024-5)

Learn More and reach out us:

Home Page: www.flow-computing.com
Email: info@flow-computing.com

FLOW