Sense- boarding algorithm in superscalar, scalable multicore and data-inspired processors.

Dr Bheemaiah, Anil Kumar, A.B Seattle W.A 98125
bheemaiaha@yopmail.com

Abstract:
A new algorithm of data dependencies and ILP is defined with the sense index of a thread in true-parallelism(™), from the definitions of Quasi-Parallelism, which is the sensitivity and sense indices defined for true scalability between single/multi-cores. The application to the CUDA architecture is delineated in formal architectural definitions.
Keywords: CUDA architectures, superscalar, ILP, data prediction, sense sensitivity index.

What:
Out of order processing in a pipeline, can be optimized with the sense-boarding processor.
In this single to multi-core scalable architecture, the processor is thread-centric with sleeping and active threads. Sleeping threads have a sense() function associated with them. Unlike their human counterparts, snoring is a useful feature that helps keep sensitive threads awake and running. sense-boarding is a scheduling algorithm that tracks the sensitivity indices of threads to snoring and helps schedule threads with dependency relationships for out of order execution.
How:
sense boarding is a board based dependency for instruction-level parallelism in multi-thread vector processing in out of order single-core/multicore symmetries.
Inter thread dependencies of data are marked in a board data-structure by maps to define sensitivity and sense indices, sense functionality is useful in the case of dependencies, resource waiting and speculative execution or in data generation and prediction. sense determines the relationship in instruction-level parallelism, to sensitive out of order and data speculation. The application to the CUDA architecture for stream processing in GPUs is also mentioned.
Algorithms are:
Instruction level parallelism in sense-sensitivity index metrics:
Data speculation , dirty caches, parallel pipeline algorithms.
Scalability in single core/ multi core implementations.
CUDA multi core architectures for stream speculation and instruction level parallelism.
Why:
Sleep is rest, and sense a measure of thread parallel-ness. While threads sleep for the right time, the awake ones perform in quasi-parallelism as HPC. Asynchronous with Lamport clocks.

Summary:

Main Points:

- Hyper-data and data flow architectures are defined as [Dc] = {[P], [MD], [S]}

- Sense-boarding is a Lamport Clock based asynchronous architecture for out of order speculative execution with data inspiration for Quasi-Parallelism.
- It is truly multi-core scalable and is data-inspired for thread interdependencies, using the sense-indices, {[I], [D]}
  ->([I], [ts], [tw], L, M)
- ILP implements naturally, scalable to vector and SIMD architectures, which are data-inspired. Extended to CUDA architectures.

Applications:
Sense Boarding leads to scalable technology, in Quasi Parallel architectures, leading to symmetric thread centric designs. Asynchronicity is emphasized, and scalable to SIMD and vector processing and also in data-centric designs in-stream and reactive architectures.

Code Base:

Introduction.

Quasi parallelism or concurrency is an asynchronous paradigm of parallel programming with multiple threads and partial dependencies between them. In quasi concurrency models, each core or many cores, simulate parallel execution in out of order scheduling of threads or blocks of code, by storing them in s queue or a similar structure, with a return to the queue of executed threads, in out of order scheduling.(Cortadella et al., n.d.; "Background on Concurrency Theory," n.d.; Saxena, Jimack, and Walkley 2018; Hoare 2010; Shiau 1996; Roosta 1999)

Problem Definition.

Quasi-Parallelism and Lamport Clocks.
We define asynchronous scheduling of threads in Quasi-parallelism using the sense boarding algorithm introduced in this paper. Time tags allow for sense boarding and speculation with indices of dependencies to preempt threads and execute in order and dependency, data dependency or speculative execution. Data flow inspired execution sequences are also described. (Cortadella et al., n.d.; "Background on Concurrency Theory," n.d.; Saxena, Jimack, and Walkley 2018; Hoare 2010; Shiau 1996; Roosta 1999)

ILP in sense-sensitivity index metrics:
{[I], [D]}
->([I], [ts], [tw], L, M)

Map from instruction execution cache [I] and closest data cache [D] to a set [I], thread collections (sleeping, executing, preemptive, retired, recycled)

We define thread dependencies on [ts], [tw] the sleeping and executing threads, and preemptive threads, sense indices, [I] and (L, M) are defined as tags on [ts], for wake

up or speculative execution. The data speculation, {[P], [MD], [S]}
Enables predicted data based speculative execution in Quasi-parallelism.

Quasi-Parallelism defines t = [b, T] where b are blocks and T a time stamp.
If Q is the queue with some blocks [b] then there is speculative execution and retirement of blocks, b_i, waiting of execution on Q, for threads [ts] and [tw] there is a similar Quasi Concurrency.
Thus we have a two-level granularity to ILP, one at the block level and another at the thread level.
Sense-sensitivity as an index, defined as a matrix is thus defined at the block and at the thread level.
(L, M) are the indices, defined in tags, represented in a binary coded form in a mapping.

<core-contribution>
The maps that define data and inter-dependencies in ILP at many granularities. If [l] is a block and [b] a thread. There exist maps defining such dependencies representable as tags storable as continuous regions of memory, denoted by (L, M) as semantic transfer or ST, a form of quantification of ontologies.
Every architecture is thus a mapping of ontologies on silicon real estate.
Thus Quasi Parallelism at the instruction, block and thread granularities are defined.
This is a topic of a future publication and more details can be found on the Github link above.
</core-contribution>

Data speculation , dirty caches, parallel pipeline algorithms.

"**Definition:** Hyper-data is defined as micro or macro hyper-scale data. In microscale hyper-data, data-mined architectures, Tr -> ([[M_i_j]], R), where QDM, is the computability of QP algorithms for data mining memory streams, for pattern clusters, metadata and sequence prediction. There can also exist DM and ZDM in P and NP for classical and stochastic algorithms, executable in [[G_i_j]]."(Bheemaiah, n.d.)

[Dc] = {[P], [MD], [S]}

Cache Coherence:
Cache policies for cache synchronization. Caches are data and instruction caches, in out of order execution, with data and interdependencies.

A cache policy for a data cache would be, Cache removal on execution or synchronization of *speculation in data prediction* in out of order synchronization.
A sense index is assigned for a binary replace in speculation alignment.
Instruction caches have a similar replacement policy, the addition of data to a cache is from a sequential stream or of instruction from a thread or
Group of threads. In the case of SIMD, we load multiple data as a tuple.

Scalability in single core/ multi core implementations.

Scalability to multicore architectures is by scheduling algorithms described in an additional publication.

CUDA multi-core architectures for stream speculation and instruction-level parallelism. sense boarding inspires a family of algorithms for data interdependency in speculative execution, with possible applicability to data-mining based architectures of micro hyperdata. This is the subject of a future publication.

Discussion.

Future Work.: Hyper-scale data and stream-based prediction of cluster patterns, metadata models and sequence prediction for data interdependency in micro hyper-scale data.

References.

"Background on Concurrency Theory." n.d. *Concurrency Theory*. https://doi.org/10.1007/1-84628-336-1_1.

Bheemaiah, Anil Kumar. n.d. "A CUDA-QUDA Architecture, Hyperscale-Data Quantum GPUs." https://doi.org/10.31224/osf.io/2q43z.

Cortadella, J., A. Kondratyev, L. Lavagno, and Y. Watanabe. n.d. "Quasi-Static Scheduling for Concurrent Architectures." *Third International Conference on Application of Concurrency to System Design, 2003. Proceedings.* https://doi.org/10.1109/csd.2003.1207697.

Hoare, Tony. 2010. "Fine-Grain Concurrency." *Concurrency and Computation: Practice and Experience*. https://doi.org/10.1002/cpe.1457.

Roosta, Seyed H. 1999. *Parallel Processing and Parallel Algorithms: Theory and Computation*. Springer Science & Business Media.

Saxena, Gaurav, Peter K. Jimack, and Mark A. Walkley. 2018. "A Quasi-Cache-Aware Model for Optimal Domain Partitioning in Parallel Geometric Multigrid." *Concurrency and Computation: Practice and Experience*. https://doi.org/10.1002/cpe.4328.

Shiau, Liejune. 1996. "Exploring Quasi-Concurrency in Introductory Computer Science." *Journal of Educational Computing Research*. https://doi.org/10.2190/7ldf-va2r-vk66-qq8d.