Advanced Architectures: Simulator release for the proposed vector extensions, and report document with performance evaluation

D4.2 v1.0

WP4 – Advanced Architectures: Simulator release for the proposed vector extensions, and report document with performance evaluation

Dissemination Level: Public

Lead Editor: Adrian Cristal

Date: 31/1/2015

Status: Final

Description of Work:

Simulator release for the proposed vector extensions, and report document with performance evaluation

Contributors:

Oscar Palomar (OP)
Adrian Cristal (AC)
Adria Armejach (AA)
Nehir Sonmez (NS)
Timothy Hayes (TH)

Internal Reviewer(s):

2ndQ: Mark Wong and Jo Dix

Version History

Version	Date	Authors	Sections Affected
0.1	22/01/2015	OP, AC, AA, NS, TH	Initial version
0.2	29/01/2015	OP, AC, AA, NS, TH	Internal revision
1.0	01/02/15	JD	final version prior to submission

Table of Contents

List of Figures

List of Tables

Abbreviations

SIMD	Single-Instruction Multiple-Data
DBMS	Database Management System
ILP	Instruction Level Parallelism
DLP	Data Level Parallelism
TLP	Thread Level Parallelism
MMX	Multimedia Extensions

SSE	Streaming SIMD Extensions
AVX	Advanced Vector Extensions
MVL	Maximum Vector Length
VSR	Vectorized Radix Sort
VPI	Vector Prior Instances
VLU	Vector Last Unique
ISA	Instruction Set Architecture
OOO	Out of Order
GPGPU	General Purpose Graphics Processing Unit

1. Summary

This document describes the simulator release for acceleration with novel vector operations (Task 4.2). Section 2.1 gives an introduction to vector architectures and motivates the approach followed in this work. Section 2.2 describes the simulator. Section 2.3 presents the experiments regarding vectorisation of hash join, Section 2.4 focuses on sorting and Section 2.5 discusses our work on aggregation. Finally, Section 3 concludes and Section 4 lists the references.

2. Analysis/ Description of work done

2.1. Introduction

Modern microprocessors often include SIMD multimedia extensions that can be used to exploit data-level parallelism. There has been some prior work using these extensions to accelerate database software, which were successful to some extent. However, multimedia extensions tend to be very limited: they often lack the flexibility to describe non-trivial data-level parallelism found in DBMS software. There is generally an upper threshold of two or four elements that can be operated on in parallel and code must be restructured and recompiled if this threshold should increase in the future. The extensions are typically targeted at multimedia and scientific application with the bulk of support geared towards floating point operations. Integer code found in DBMS software was not its intended purpose, this can be a limiting factor to exploit the parallelism in such applications.

2.1.1 Vector Architectures

Moore's Law [M+65] famously predicted that the number of transistors in an integrated circuit will double approximately every two years. Based on Moore's Law, similar observations related to performance were also made, such as Dennard scaling [DGR+74] and Koomey's law [KBSW11]. These observations were heavily reliant on the frequency scaling of microprocessors. For a long time it was reasonable to shrink the size of a transistor and expect it to have a lower latency and thus higher operating frequency. Two main problems challenged these trends. The first was that the gap between the performance of the microprocessor and the main memory grew drastically. The second is that thermal and power issues eventually made it infeasible to scale the processor's frequency any longer. The free performance scaling that we had grown accustomed to finally came to an end [Sut05]. From that moment on, the industry has had to rely on parallel techniques in order to gain more performance from their workloads.

Parallel techniques can be broadly categorised as instruction-level (ILP), thread-level (TLP) and data-level (DLP). When it is possible to exploit it, DLP is by the far the most efficient form of parallelism [HP12]. Typically DLP is defined as repeatedly performing the same task on many data inputs; stricter definitions may require the data to be independent whereas more lax definitions may be extended to include semi-independent data. One of DLP's principal advantages lies in the fact that many operations can be expressed in an ultracompact form with much less encoding overhead over ILP and TLP techniques. This compact form in itself can be a major source of potential speedup and also allows for efficient hardware implementations that can infer and exploit the independence of the data and the repetitiveness of the operation. Some former research in DBMSs has shown that various operations do contain some DLP [ZR, WPB+ 09, Mar96, MK00, HNZB07, HLY+09, CNL+ 08, KKL+ 09]. All of this research, however, was done outside the sphere of the computer architecture domain. This is important because it implies that researchers were working with a limited set of tools in the form of ISAs and hardware and tried to solve their problem within these constraints. We feel that looking at this area from a different angle could be fruitful. What needs to happen to algorithms, instruction sets and hardware in order to effectively exploit data-level parallelism in DBMSs? There is potential to discover new algorithms, previously impossible to implement, by defining new instructions as well as optimising existing algorithms by tailoring the hardware to the available DLP.

DLP can be exploited in numerous ways. Recent developments in microprocessor architectures have pushed a focus on multi-core acceleration. While this is an effective technique to exploit thread-level parallelism, single-instruction multiple-data (SIMD) instruction sets offer a more efficient way to accelerate DLP [LAB+ 11]. To clarify, we use the term SIMD to encapsulate many different hardware techniques, e.g. vector processors, multimedia ISA extensions, general purpose graphics processing units (GPGPUs). There has typically been some level of support for SIMD instructions in general-purpose microprocessors, commonly dubbed multimedia extensions, e.g. MAX-1, MMX, Altivec and SSE. There has been some prior work [ZR, WPB+ 09] using these multimedia extensions to accelerate database software, however these extensions are typically targeted at multimedia and scientific applications with the bulk of support geared towards floating point operations. Integer code found in DBMS software was not their intended purpose, this can be a limiting factor in exploiting the parallelism in such applications. Furthermore, in typical multimedia extension implementations the data is expected to have high spatial locality which is typically found in multimedia applications but not as much in business-domain applications [BGB].

Multimedia ISA extensions also offer many advantages over dedicated accelerators such as GPGPUs [OLG+ 07]. One major advantage is that the extensions share the same memory hierarchy as the central processor. Dedicated accelerators generally have their own memory hierarchies and need to move data structures to and from each device in order to operate on them. Another major advantage is that multimedia extensions can be tightly coupled into the core, e.g. SIMD functional units can be placed on the main pipeline and benefit from modern optimisations such as superscalar and out of order execution, a low latency cache, register renaming and branch prediction. These two advantages are ideal when executing an application that has a fine-grained interlacing of scalar and SIMD code that interact and rely heavily on one another. Although these multimedia ISA extensions started out relatively simple, successive generations have become more sophisticated and offer wider SIMD registers to process more elements per instruction as well as more intricate instructions to operate on them. AVX-512 [Int14] will increase the width of the registers to 512 bits and include mask registers, full gather/scatter support and many non-trivial SIMD instructions; initial implementations are expected in 2015 for Knights Landing (Intel's next-generation Xeon Phi processor for high-performance computing). This trend is anticipated to continue in the future and the SIMD register width and instruction sets are expected to grow further. We therefore predict that the SIMD support found in commodity microprocessors will eventually resemble the instruction sets of classic vector architectures traditionally found in supercomputers [EVS98].

Vector architectures are highly scalable and overcome the limitations imposed by SIMD multimedia extensions. They have the flexibility to represent complex DLP which multimedia extensions lack. The registers of a vector architecture typically hold more elements than those of the multimedia extensions. Furthermore, it has been typical for multimedia extensions to have counterpart functional units as wide as their register, i.e. the number of lanes is equal to the number of elements in the multimedia register. This is in contrast to vector architectures which have traditionally pipelined their operations through narrow functional units or opted for a compromise between the two extremes with a number of lanes larger than one but smaller than the number of elements in a vector register.

Vector architectures have been around since the early 1970s and were traditionally used in the supercomputer market [Rus78, HP71, EVS98, Sch87]. Vector architectures have traditionally been used for scientific applications abundant with floating-point code, however their applicability to business domain (integer) applications has yet to be analysed. They provide sophisticated ISAs to express data-level parallelism concisely. The hardware is typically much more simple than aggressive out-of-order superscalar microarchitectures yet can provide strong computational performance when the application is suitable (vectorisable).

Vector architectures are energy-efficient [LSCJ06, LAB+ 11] and can be implemented in microprocessors using simple and efficient hardware [Asa98]. There can be a large reduction in the pipeline's fetch and decode stages and scalability is generally not inhibited by associative hardware structure as with out-of-order superscalar microarchitectures [PJS97]. Additionally, there is less requirement for aggressive speculation and memory prefetching since execution and access patterns are fully encoded into a single instruction.

Historically, vector architectures fully pipelined operations through its functional units to hide execution latency. SIMD multimedia extensions are instead executed using very wide functional units. Vector architectures have often employed chaining to pipe the output of one instruction into the input of another instruction before the former fully completes. SIMD multimedia extensions do not do this because they do not fully pipeline their inputs.

Vector architectures have often included a separate set of architectural registers to enable masking. Masking allows turning conditional branches into predicated instructions. This can help eliminate bottlenecks due to branch mispredictions. The SIMD multimedia extensions have some support for masking, although not through separate registers, but it is very limited. The most lacking feature in the SIMD multimedia extensions' idea of masking is that it cannot conditionally execute parts of a memory operation. This has changed recently in AVX which allows a limited form of masked load/store instructions. Some vector architectures included support for reductions instructions. These allow a vector of values to be aggregated into a single scalar value. The SIMD multimedia extensions offer no equivalent to this. Vector architectures have more options when it comes to memory instructions. Vector machines typically support unit-stride memory accesses, strided memory accesses and indexed memory accesses. The SIMD multimedia extensions require that all data be stored in contiguous locations in memory. Vector architectures have usually included more flexible instructions than those found in the SIMD multimedia extensions. For example, the compress instruction found in many vector architectures [SFS00] allows an input vector register to be rearranged based on a mask. The SIMD multimedia extensions have some position manipulation instructions, but are generally not as powerful as those in their vector counterparts.

One the strongest advantages of vector processors is their ability to tolerate long latency instructions, especially memory operations. In contrast, the SIMD multimedia extensions have a much more limited access to memory. Another advantage is their ability to take advantage of the entire memory bandwidth available. As an example, Sandy Bridge [Int14a], a microarchitecture that implements AVX, provides a 16-byte bus from the SIMD registers to the level 1 cache. If there is a cache miss, the instruction is treated the same as scalar load that misses cache. The benefit is that the bandwidth is doubled, but only if the working set is cache resident. As memory latency and bandwidth has become a significant issue for both computer architects and software developers, vector support could be instrumental in optimising databases which are typically bottlenecked by the memory [BMK].

This work takes a top-down methodology and profiles PostgreSQL looking for opportunities to use vector technology. From this, software bottlenecks caused by unscalable scalar hardware structures are identified and new vector ISA extensions for x86-64 are proposed as a solution. These ISA extensions are currently being implemented using a cycle-accurate microprocessor simulator.

2.1.2 Methodology

The methodology of this work determines the manner in which the objectives can be achieved. Here we decide a general strategy of the thesis as well as choosing particular details to accomplish our objective. In order to study the acceleration with novel vector operations, we need to rely heavily on hardware simulation. We give the following reasons. The SIMD capabilities in x86-based microprocessors have moved from simple 64-bit multimedia extensions in MMX to 128 bits in SSE, and to registers from 256 bits to 512 bits in AVX and AVX-512 respectively. This trend is expected to continue for the foreseeable future thus it is imperative to understand what the performance consequences will be when running vectorisable algorithms on future microprocessors with wider SIMD registers. This work aims to explore the performance characteristics of DLP-accelerated sorting algorithms when the SIMD register width is as large as 2,048 bits (possibly larger).

In both the Xeon Phi and AVX2 implementations, SIMD instructions are issued to functional units that are as wide as the SIMD registers, i.e. all elements of the operation are processed in parallel. This is in contrast to a traditional vector architecture like the Cray-1 [Rus78] which instead pipelined its SIMD instructions. We suspect that there are algorithms that can be accelerated with SIMD instructions that are implemented in a pipeline and will show little extra benefit when using lock-stepped parallel lanes. It is therefore important to be able to experiment with this parallel factor in order to better understand the algorithms' characteristics.

The Xeon Phi offers a more sophisticated SIMD ISA than AVX2 and includes useful instructions such as gather/scatter as well as masked operations. One drawback is that its microarchitecture uses a simple dual-issue in-order pipeline based on the original Pentium [Chr12, SCS+ 08]. Prior work [EVS97] shows that there are advantages when executing SIMD instructions with an out-of-order superscalar microarchitecture. It is also felt that to make a fair comparison to scalar algorithms a similar, if not equal, baseline microarchitecture should be used. Our goals are to look at the performance characteristics of existing vectorised algorithms running on future microarchitectures as well as proposing new instructions and hardware to facilitate new algorithms.

Achieving these goals would be impossible if using only existing architectures, therefore we have created a simulation environment in order to make the necessary experiments. Because a simulation environment is several orders of magnitude slower than a real processor we don't expect to fully simulate a full DBMS implementation nor a full run of TPC-H. Instead, we will create microkernels with behaviour and datasets inspired by our observations in the characterisation stage. The microkernels should be much lighter than a full DBMS implementation but at the same time be representative of the workload. One important decision we have made is not to rely on a vectorising compiler. This is due to the fact that current open source vectorising compilers are not up to scratch for the level of quality we expect in our DLP transformations. More often than not, DBMS software is written in C and C++ and even the best vectorising compilers have difficulty working with these languages due to their strict semantics and potential memory aliasing side effects. Involving a compiler would also put strain on the research to modify and support it which goes outside the scope of our research. Instead we intend to vectorise our microkernels by hand with our own ISA. This way we get full control over the transformation and can ensure optimal quality in all the code. We must be careful when comparing with scalar baselines that we also ensure that these scalar kernels are also performing at their peak.

2.2. Description of the simulator release

Deliverable D4.1 presented a detailed description of the proposed vector extensions and their implementation in the simulator. In order to prevent this document growing unnecessarily, we reproduce here only the most relevant information and refer to D4.1 for more details.

At its heart we use PTLsim [Your07], a cycle-accurate x86-64 simulator. PTLsim models many features of modern OoO superscalar processors including: µop translation, multistage pipelines, speculation and recovery, and a multi-tiered cache hierarchy. We have configured the simulator to behave as closely as possible to Intel's Westmere microarchitecture [Int14a]. Table I contains the various microarchitectural parameters. There are six execution unit clusters in total: a load address generation cluster; a store address generation cluster; a store data cluster; and three arithmetic (non-memory) clusters.

Table I: Microarchitecture parameters

a. superscalar and out-of-order

parameter	**value**	**parameter**	**value**
fetch width	4	**fetch queue**	28
frontend width	4	**frontend stages**	17
dispatch width	4	**writeback width**	4
commit width	4	**reorder buffer**	128
issue width per cluster	1	**total issue width**	6

issue queue per cluster	8	**total issue queue**	48
load queue	48	**store queue**	32
L1-d misses	10	**L2 misses**	16

b. cache hierarchy

level	**size**	**latency**	**line size**	**ways**	**sets**
L1-i	32 KB	1	64	4	128
L1-d	32 KB	4	64	8	64
L2	256 KB	10	64	8	512

By default PTLsim uses a fixed latency memory system which does not model bandwidth and contention issues. Recent work on vector processors [HPUCV12] has shown that they have the ability to saturate a system's available bandwidth, thus making it crucial to model the memory system accurately when executing vectorised algorithms, otherwise the results may be inaccurate and misleading. For this reason we have integrated DRAMSim2 [RCJ11], a cycle-accurate memory system simulator, into PTLsim and replaced the default memory model. Having an accurate memory model allows the vectorised algorithms to work within a realistic bandwidth envelope and thus enforces a fairer comparison to non-vectorised algorithms. Table II shows the memory system parameters. The simulated processor has a frequency of 2.67 GHz and as a result the memory controller is clocked every four processor cycles. Additionally, the following address layout scheme is used as it was found to work well with all of our experiments: row:rank:bank:column:burst.

Table II: Memory system parameters

parameter	**value**	**parameter**	**value**
type	DDR3-1333	**transaction queue**	64
clock	1.5 ns	**command queue**	256
policy	open page	**row accesses**	8
queue	per rank per bank	**banks**	8
scheduling	rank then bank	**ranks**	4
rows	32,768	**columns**	2,048

burst length	64 bytes	**device width**	4

1) Baseline SIMD Support: We have modified the simulation framework substantially in order to give it extensive SIMD support. All instructions listed here can be commonly found in conventional vector ISAs. For brevity, we give a high-level overview that captures the most important features of these SIMD additions.

We have extended the x86-64 ISA with sixteen logical vector registers and four logical mask registers. The width of these registers is a configurable parameter of the simulator in order to experiment with different maximum vector lengths (MVL). The MVL determines the number of 64-bit elements per vector register. As the baseline microarchitecture uses register renaming, we also apply this technique to the new vector registers. It has been shown that renaming vector registers is beneficial when the number of physical registers is double the number of logical registers [EVS97], hence we provide thirty-two physical vector registers and eight physical mask registers. There is also an additional vector length register that controls the number of elements operated on by any given vector instruction. This is explicitly managed with get/set instructions.

The vector capabilities are tightly integrated into the microarchitecture. Execution of vector instructions is performed out-of-order (see D4.1). We have added three new clusters: one to perform the address generation of vector memory instructions and two to execute non-memory vector instructions.

We have defined and implemented new vector memory instructions. Three patterns are supported, each with its own load, store and prefetch instructions: (1) unit-stride: memory is accessed contiguously. This is the most efficient access pattern due to the spatial locality of the elements accessed. (2) strided: memory instructions use a base address and a parameter that refers to the increment in memory between elements which are adjacent in the vector register. (3) indexed: also known as gather/scatter, the instructions use a base address and an additional vector register of offsets.

Unit-stride and strided instructions calculate their addresses formulaically and the number of cycles spent performing the address generation depends on the number of cache lines needed to fulfil the request, e.g. four cache lines accessed would require four cycles in the functional unit. Indexed memory instructions require adding an offset to a base address and require VL/lanes cycles to perform address generation where VL is the vector length of the instruction. Using an existing technique [HPUCV12], [EAEF02], [QCEV99], the vector register file bypasses the L1-d cache and goes directly to the L2 cache. This way more bandwidth can be provided at the expense of higher latency; it has been shown in the cited work that this extra latency is easily amortised due the large number of elements operated on per individual instruction. The number of cycles needed to complete a memory instruction depends on how many individual cache lines are requested and whether or not these are already resident in the L2 cache. In order to simplify the implementation of vector memory instructions, the hardware allows them to execute out-of-order, without checking for memory aliasing. Instead, vector memory fence instructions have been introduced. A fence instruction enforces all subsequent memory instructions of one type (vector, scalar) to wait until the older memory instructions commit. Thus, the programmer or the compiler uses the fences to explicitly indicate when there is a risk of memory ordering violation and ensure correctness of computation. Vector memory fences are explained in more detail in D4.1 and [HPUCV12].

To operate on the vector registers, we have added a suite of non-memory vector instructions which is summarised in Table III. Each instruction requires VL/lanes cycles to pass through a functional unit, with the exception of mask instructions which require one cycle. Comparison instructions produce a result to a mask register. Most instructions can be optionally masked with the exception of the permutative instructions, which rearrange the order of the input vector's elements. merge and iota are instructions found in the CRAY-1 [Rus78].

Table III: Non-memory vector instructions

class	**instructions**
initialisation	set all, clear all, iota
arithmetic	integer add, subtract, multiply
bitwise logical	and, or, not, xor, shift left, shift right
comparison	greater than, less than, equal
mask	set, clear, and, or, not, xor, popcount
permutative	compress, shuffle, reverse
other	merge, copy, get/set element
vector length	get, set, set MVL

2) Novel vector instructions:
The following instructions have been added to the baseline vector ISA. They are not found in conventional vector ISAs. For this reason, they are described in more detail than the rest of instructions.

prior instances (vpi):

semantics:

```
for (i=0; i<vlen; i++)
```

```
{
  if ( (!mask) || (mask && mask[i]) )
  {
     vrd[i]=0;
     for(j=0; j<i; j++)
     {
        if (!mask) || (mask && mask[j]) )
        {
           if(vra[i] == vra[j])
           {
              vrd[i]++;
           }
        }
     }
  }
}
```

This will count the number of times a value has occurred previously in a vector register.

last unique (vlu)

semantics

```
for (i=vlen-1; i>=0; i--)
{
  if ( (!mask) || (mask && mask[i]) )
  {
     found=false;
     for(j=i+1; j<vlen; j++)
     {
        if ( (!mask) || (mask && mask[j]) )
        {
           if (vra[i] == vra[j]) found=true;
        }
     }
     maskd[i] = !found;
  }
}
```

comments

This will mark the last instance that a unique or repeated value appears in a vector register.

multi-reduce

semantics

```
for (i=0; i<vlen; i++)
{
  if ( (!mask) || (mask && mask[i]) )
  {
     vrd[i]=0;
     for(j=0; j<=i; j++)
     {
        if (!mask) || (mask && mask[j]) )
        {
           if(vra[i] == vra[j])
           {
              vrd[i] += vrb[j];
           }
        }
     }
  }
}
```

comments

This will perform sum reductions for each group.
Instead of outputting a total end value, each element is the partial sum until that moment of processing.

intradelta

semantics:

```
for (i=0; i<vlen; i++)
{
  if (i==0)
  {
    maskd[i] = (vra[i] != sra)
  }
  else
  {
    maskd[i] = (vra[i] != vra[i-1])
    }
}
```

memory fences

fence: scalar store to vector memory operation (fssvm)
Commits all prior scalar stores before any subsequent vector memory operation can issue.

fence: vector store to scalar memory operation (fvssm)
Commits all prior vector stores before any subsequent scalar memory operation can issue.

fence: vector store to vector memory operation (fvsvm)
Commits all prior vector stores before any subsequent vector memory operation can issue.

2.3. Hash join

We first characterised a state of the art decision support DBMS called Vectorwise [Act]. The main reason to use Vectorwise is that we required a column-stored DBMS in order to efficiently vectorise the DBMS. We profiled this application with the TPC-H decision support benchmark [Tra11] and discovered that the hash join operation accounts for 61% of its execution time. We demonstrated that this operation had disproportionate performance returns when scaling/increasing out-of-order and superscalar parameters in a cycle-accurate simulator modelling the Intel Nehalem microarchitecture. We then took the most significant part of the operation, the probe phase, and found a lot of data-level parallelism that couldn't be captured by existing SIMD multimedia extensions.

Based on these observations, we designed our own vector extensions for a modern out-of-order superscalar x86-64 commodity microprocessor. These extensions were inspired by the ISAs of classic vector supercomputers but were tailored and optimised for DBMS support. We implemented these extensions with a custom simulation framework based on PTLsim [Your07] and DRAMsim2 [RCJ11]. Our results show significant speedups with good scalability for medium to long vector register lengths and future memory bandwidths.

The following figure shows how speedup over scalar baseline grows with the maximum vector length (MVL), for several input data sizes. In the figure, *tpch* dataset queries a hash table of 32 MB extracted from TPC-H query 9. Four extra synthetic datasets have been added in order to evaluate the algorithm in different scenarios. *l1r* and *l2r* are built such that the data structures used can be resident in the L1D and L2 caches respectively. The left hand side input does not have temporal locality and is therefore not considered in this measurement. *huge* has structures of an equal size to tpch but with a different selectivity (the left hand side finds more matches in the right hand side) leading to more work. *2mb* has an intermediate size between l2r and huge.

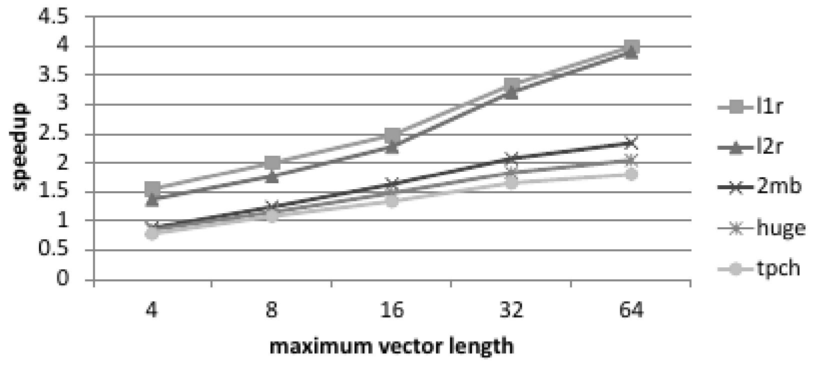

Figure 1: The speedup over scalar baseline

The following figure shows the impact in speedup of increasing memory bandwidth (one memory controller (mc1), two memory controllers (mc2) and infinite bandwidth (inf. bw). Longer vectors can leverage the available bandwidth to a greater extent and yield more performance. Please note that the scalar version does not show any noticeable improvement from increasing the memory bandwidth. For more details, please see [HPUCV12].

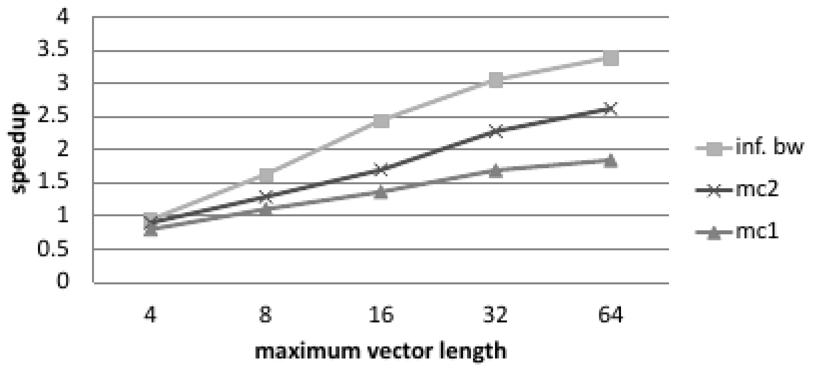

Figure 2: The impact in speedup of increasing memory bandwidth

2.4. Sorting

We have made a very large contribution in the area of vectorised sorting. We implemented three existing vectorisable algorithms to run with our vector extensions: quicksort, bitonic mergesort and radix sort. We showed the strengths, weaknesses and scalabilities of each algorithm when run on our simulation framework. Based on these findings we then proposed our own vectorised sorting algorithm: VSR sort. VSR sort is inspired by radix sort, however the algorithm circumvents the drawbacks identified in the previous vectorised version of the algorithm. To enable this algorithm we have had to propose new vector instructions as well as complementary hardware: VPI and VLU. The idea of these new instructions is to identify and rectify gather/scatter conflicts that may occur in the new algorithm. The following figure summarises the results for the three reference algorithms and VSR, for vector extensions of MVL from 8 to 64.

They are evaluated with three different data sets (small, medium, large). The figure shows speedup over scalar baseline. We found that VSR sort has very significant speedups over all prior work. Please, see deliverable D1.6 and [HPUCV15] for more details.

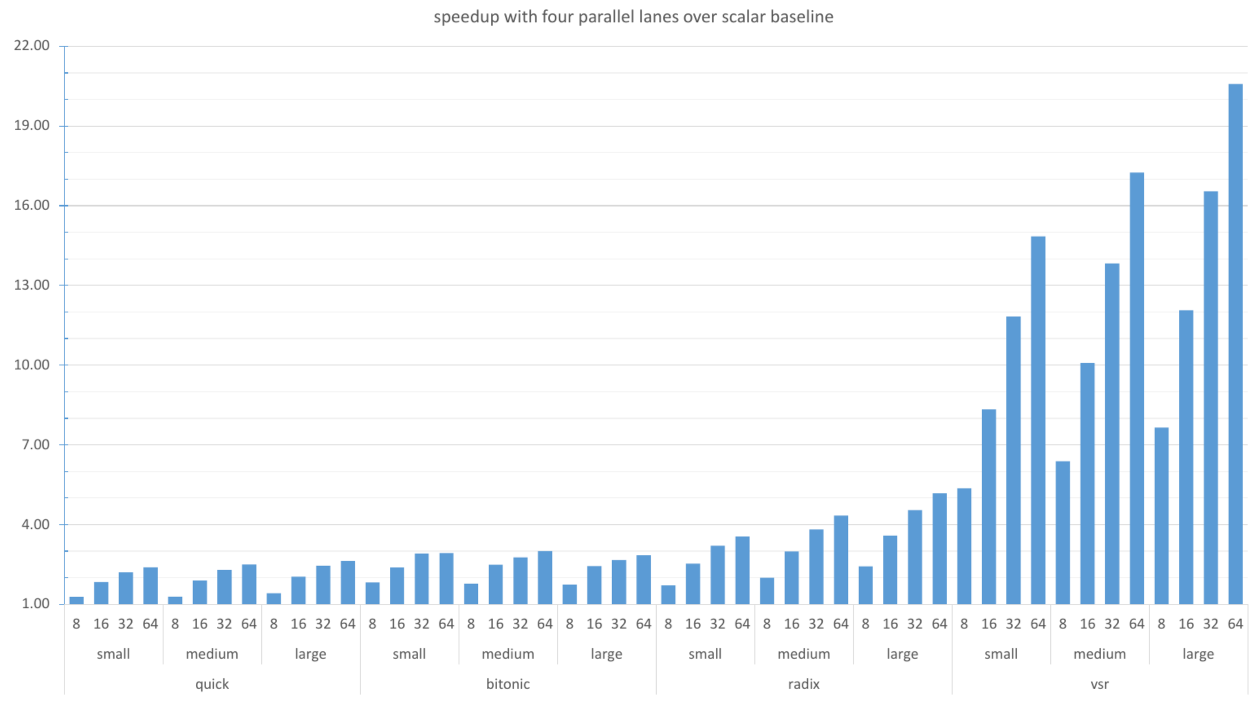

Figure 3: The sorting speedup over scalar baseline

2.5. Aggregation

The goal of this work is to explore different vectorisation techniques applied to reduction and aggregation operations.

Aggregation, i.e. vector reductions, are extremely important in decision support databases. Aggregation is an important operation in DBMS systems. In TPC-H it dominates queries 1, 4, 9, 13, 15, and 18. Aggregations can be non-trivial to implement due to the GROUP BY statement. There is no 'best-fit' solution for all scenarios due to the different behaviour and characteristics of aggregation, e.g. reduction size. For example: TPC-H Q1 has six independent aggregations with few groups whereas TPC-H Q18 has one aggregation with many thousands of groups.

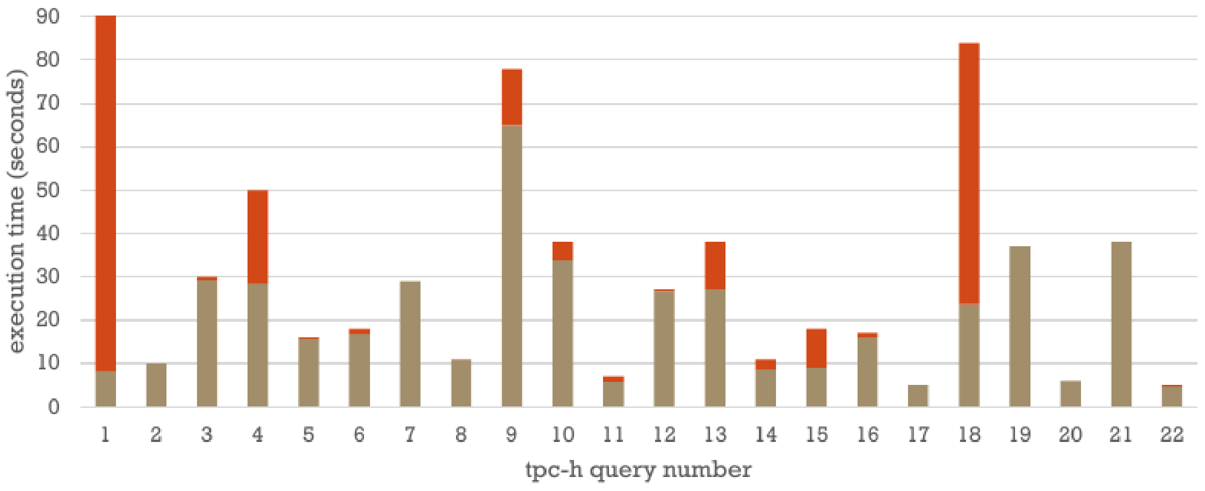

Figure 4: Aggregation in TPC-H

Scalar techniques have used hash tables to implement this operation. If a hash table is implemented strictly as a histogram (i.e. no conflicts allowed) then vectors could also use this technique provided histogram localisation is used. Our work on sorting showed that while localised histograms can enable vectorisation, performance can be poor due to cache locality problems. In the same work we proposed conflict-correcting vector operations specifically to enable radix sort. Unfortunately these are not completely suitable for aggregation as they only help incrementing a histogram by 1 rather than by an auxiliary number. It should be researched/explored first, but the addition of this non-unit number complicates the process. Additionally, it may not help much. For example, with TPC-H Q1 there is not going to be a problem with replicated histograms because of the small number of groups, and with Q18 there is going to a problem with or without the replication due to the larger number of groups.

A more promising solution would be to leverage our VSR sort, thus indirectly exploiting conflict-correcting vector operations, in order to sort the input and after reduce it. VSR sort would be perfect because it is stable and therefore is suitable for multi-column indices. Once the input is sorted, a segment mask can be created to mark the partitions. After, this mask can be used with a new kind of segment scan [Ble89] to do the reductions. TPC-H Q1 can benefit from having a single segment mask used for many reductions.

A challenge lies in the implementation of the segment scan. In TPC-H Q1 we will have long segments that will require strip mining with many iterations. With Q18 we will get short segments with an average length of four elements. We need to be able to exploit both intra- and inter-segment parallelism without overly complicated logic. We will compare this with a scalar version as well as a vector version that uses histogram localisation. This problem is interesting because although sorting a key can be more expensive than hashing it, the post-sort state of input allows for new and interesting DLP techniques.

2.5.1 Aggregate Functions

To work with the vector hardware the reduction operation must be associative. Typical aggregate functions are listed here.

AVG() - Returns the average value
COUNT() - Returns the number of rows
FIRST() - Returns the first value
LAST() - Returns the last value
MAX() - Returns the largest value
MIN() - Returns the smallest value
SUM() - Returns the sum

In TPC-H, we encounter only SUM(), COUNT() and AVG(). AVG() can be considered a combination of SUM() and COUNT(). TPC-H also selects the first N rows of a result, however this doesn't really seem to be the same as FIRST() which is a vectors⇒scalar reduction.

2.5.2 Existing vector solutions

Vectors and aggregations do not work well together. Vectors are optimised for vertical operations whereas aggregations are inherently horizontal. Adding in GROUP BY, i.e. binned, aggregations makes this even more complex since it decreases the straight-forwardedness of the operation and introduces the possibility of conflicts for some vector solutions. As far as we know, there is no published research on DBMS aggregations in the context of a vector processor however they have cropped up three times in the context of SIMD: [ZR], [HNZB07], [POR12].

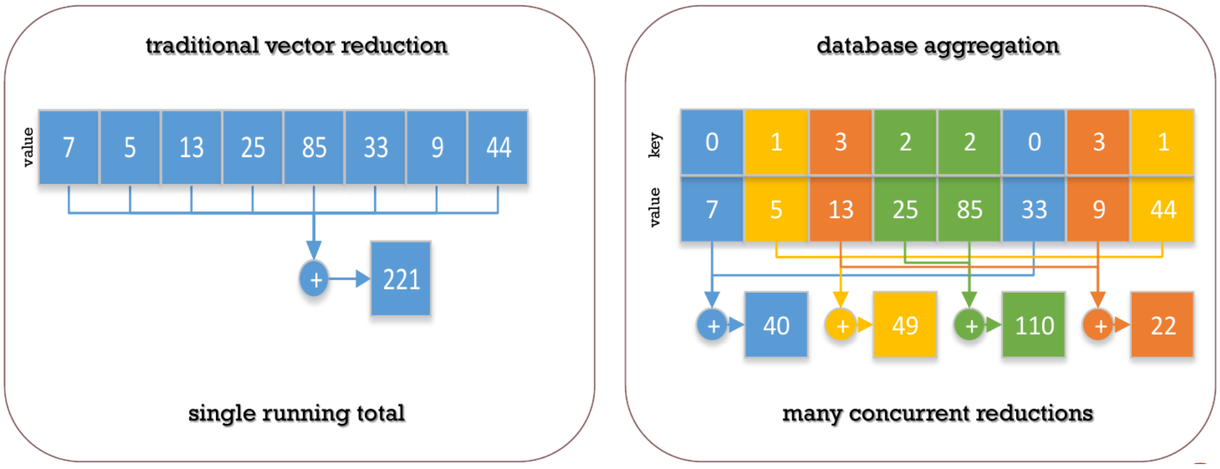

Figure 5: Vector reduction vs DB aggregation

There doesn't seem to be an obvious algorithm for a vector processor with a rich ISA. The solutions that we have found may work fine in some scenarios and poorly in others. We outline two solutions. The first when the GROUP BY key(s) are not sorted so we use a hash table approach. The second when the GROUP BY key(s) are sorted so we use regular reduction instructions with some overhead to calculate lengths and do strip mining.

Vector reductions are too simple for database aggregations. They use an additional vector of keys and reduce a vector to multiple scalar values based on key. Moreover, there is no one-size-fits-all solution and the best solution depends on the query, e.g.
TPC-H query 1

- 6x value columns
- 2x key columns
- Columns not presorted
- Reduces to 4 groups
- Hashing works best

TPC-H query 18

- 1x value column
- 1x key column
- Columns presorted
- Reduces to 1,500,000 groups
- Sorted reduce works best

2.5.3 Proposed solution

Our novel vector technique is suitable for both TPC-H queries 1 and 18. It leverages VSR (see Section 5) to sort the input very quickly and then uses two new instructions to perform the reduction in a vectorised fashion to the sorted input. Intradelta is used to find markers between groups of consecutive elements, and multi-reduce reduces many groups inside a vector with a single instruction.

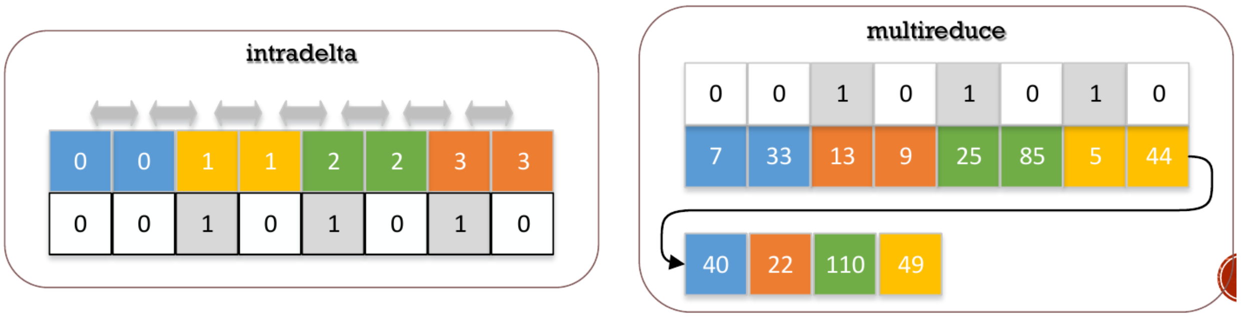

Figure 6: Intradelta and multireduce

The preliminary results, shown in the next two figures, indicate that there is significant advantage in vectorising aggregation in Q1 and Q18. Results are compared against two scalar implementations (hash based and sort based). In Q18, results for hash based scalar version are much worse than the rest so they are not included. Speedup is reported over the scalar version with best results for the particular query (sort in Q1, hash in Q18). Speedup increases significantly with the vector length.

Q1:

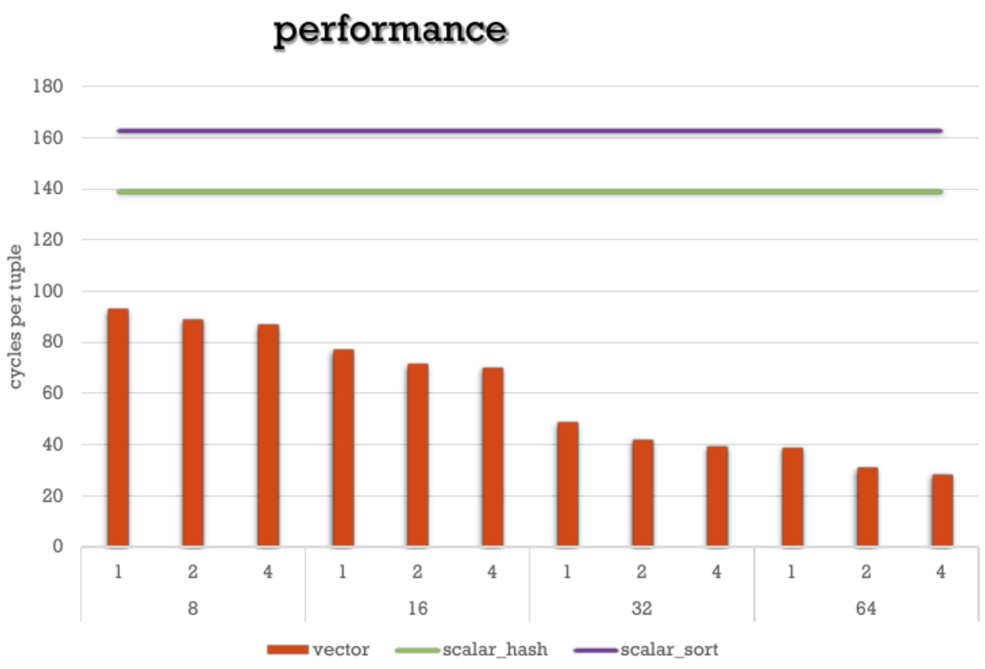

Figure 7: Q1 performance

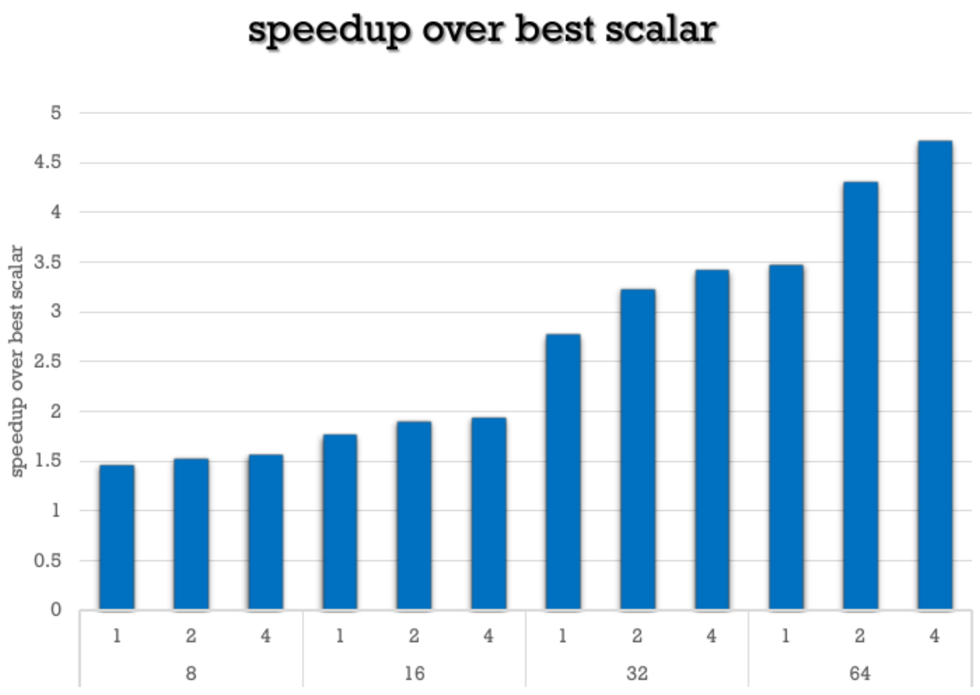

Figure 8: Q1 speedup

Q18:

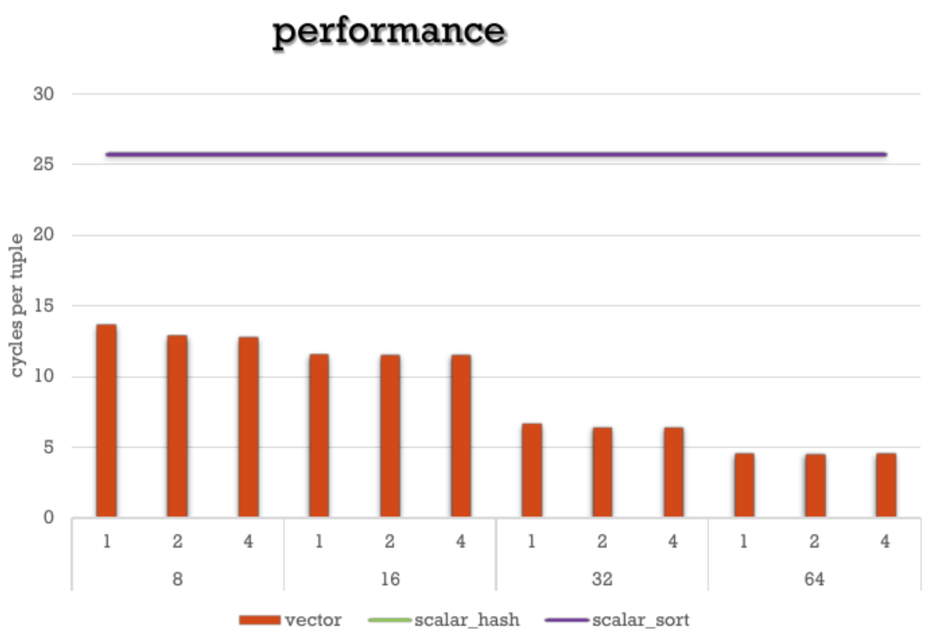

Figure 9: Q18 performance

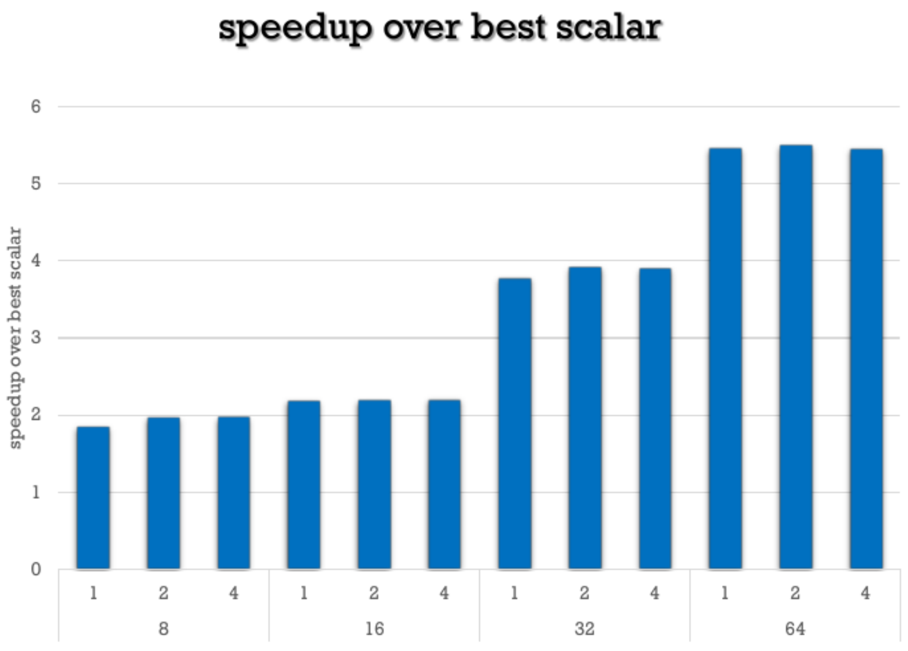

Figure 10: Q18 speedup

3. Conclusions / Future work

We have developed a detailed simulator that implements a complete vector extension compatible with the Intel x86 ISA. The simulator has been integrated with an accurate DRAM simulator in order to model memory appropriately. As our experiments show, memory bandwidth has a huge impact on performance of vector architectures and should be simulated accurately. The baseline vector ISA, that contains instructions commonly found in conventional vector processors, has been extended with several novel vector instructions that we have proposed after careful analysis of the benchmarks. This document describes the semantics of all these instructions.

The simulator has been used to study the effect of vectorisation on three important algorithms/parts of DBMSs: hash join, sorting and aggregation. Results for all of them show high speedup over carefully optimised scalar versions, when the novel vector extensions are used. A novel vectorised algorithm for sorting, based on radix sort, has been proposed.

The work on hash join and sorting has been completed, while work on aggregation is still preliminary. Future work will include several sensitivity analyses, as well as a power model to analyse the impact on power and energy consumption when including the vector extensions we proposed in this document.

4. References

[Act] Actian. Vectorwise. Record Breaking Action Engine for Big Data. http://www.actian.com/products/vectorwise.
[Asa98] Krste Asanović. Vector Microprocessors. PhD thesis, EECS Department, University of California, Berkeley, 1998.
[BGB] Luiz André Barroso, Kourosh Gharachorloo, and Edouard Bugnion. Memory system characterization of commercial workloads. In ISCA ’98, pages 3–14.
[Ble89] Guy E Blelloch. Scans as primitive parallel operations. Computers, IEEE Transactions on, 38(11):1526–1538, 1989.
[BMK] Peter A. Boncz, Stefan Manegold, and Martin L. Kersten. Database architecture optimized for the new bottleneck: Memory access. In VLDB ’99, pages 54–65.
[Chr12] George Chrysos. Intel ® Xeon PhiTM coprocessor (codename Knights Corner). In Proceedings of the 24th Hot Chips Symposium, HC 2012, 2012.
[CNL+ 08] Jatin Chhugani, Anthony D. Nguyen, Victor W. Lee, William Macy, Mostafa Hagog, Yen-Kuang Chen, Akram Baransi, Sanjeev Kumar, and Pradeep Dubey. Efficient Implementation of Sorting on Multi-Core SIMD CPU Architecture. Proceedings of the VLDB Endowment, 1(2):1313–1324, August 2008.
[DGR+74]Robert H Dennard, Fritz H Gaensslen, V Leo Rideout, Ernest Bassous, and Andre R LeBlanc. Design of ion-implanted mosfet’s with very small physical dimensions. Solid-State Circuits, IEEE Journal of, 9(5):256–268, 1974.
[EAEF02] R. Espasa, F. Ardanaz, J. Emer, S. Felix, J. Gago, R. Gramunt, I. Hernandez, T. Juan, G. Lowney, M. Mattina, and A. Seznec, “Tarantula: A Vector Extension to the Alpha

Architecture," in Proceedings of the 29th Annual International Symposium on Computer Architecture, ser. ISCA, 2002, pp. 281–292.
[EVS97] R. Espasa, M. Valero, and J. E. Smith, "Out-of-Order Vector Architectures," in Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture, ser. MICRO, 1997, pp. 160–170.
[EVS98] Roger Espasa, Mateo Valero, and James E. Smith. Vector Architectures: Past, Present and Future. In Proceedings of the 12th International Conference on Supercomputing, ICS '98, pages 425–432, 1998.
[HLY+ 09] Bingsheng He, Mian Lu, Ke Yang, Rui Fang, Naga K. Govindaraju, Qiong Luo, and Pedro V. Sander. Relational Query Coprocessing on Graphics Processors. ACM Transactions on Database Systems, 34(4):21:1–21:39, 2009.
[HNZB07] Sándor Héman, Niels Nes, Marcin Zukowski, and Peter Boncz. Vectorized Data Processing on the Cell Broadband Engine. In Proceedings of the 3rd International Workshop on Data Management on New Hardware, pages 4:1–4:6, 2007.
[HP71] SA Holland and CJ Purcell. The cdc star-100 a large scale network oriented computer system. ieee Internat. In Comput. Soc. Conf, pages 55–56, 1971.
[HP12] John L Hennessy and David A Patterson. Computer architecture: a quantitative approach. Elsevier, 2012.
[HPUCV12] T. Hayes, O. Palomar, O. Unsal, A. Cristal, and M. Valero, "Vector Extensions for Decision Support DBMS Acceleration," in Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO, 2012, pp. 166–176.
[HPUCV15] Timothy Hayes, Oscar Palomar, Osman Unsal, Adrian Cristal, Mateo Valero (Barcelona Supercomputing Center), "VSR Sort: A Novel Vectorised Sorting Algorithm and Architecture Extensions for Future Microprocessors", to appear in HPCA 2015.
[Int14]Intel. Intel Architecture ® Instruction Set Extensions Programming Reference, March 2014.
[Int14a] Intel 64® and IA-32 Architectures Optimization Reference Manual, Intel, March 2014.
[KBSW11] J.G. Koomey, S. Berard, M. Sanchez, and H. Wong. Implications of historical trends in the electrical efficiency of computing. Annals of the History of Computing, IEEE, 33(3):46–54, March 2011.
[KKL+ 09] Changkyu Kim, Tim Kaldewey, Victor W. Lee, Eric Sedlar, Anthony D. Nguyen, Nadathur Satish, Jatin Chhugani, Andrea Di Blas, and Pradeep Dubey. Sort vs. Hash Revisited: Fast Join Implementation on Modern Multi-Core CPUs. Proceedings of The VLDB Endowment, 2(2):1378–1389, 2009.
[LAB+ 11] Yunsup Lee, Rimas Avizienis, Alex Bishara, Richard Xia, Derek Lockhart, Christopher Batten, and Krste Asanovi̍ c. Exploring the Tradeoffs Between Programmability and Efficiency in Data-parallel Accelerators. In Proceedings of the 38th Annual International Symposium on Computer Architecture, ISCA '11, pages 129–140, 2011.
[LSCJ06] Christophe Lemuet, Jack Sampson, Jean-Francois Collard, and Norm Jouppi. The Potential Energy Efficiency of Vector Acceleration. In Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, 2006.
[M+65] Gordon E Moore et al. Cramming more components onto integrated circuits, 1965.
[Mar96] Rich Martin. A Vectorized Hash-Join. IRAM technical report, University of California at

Berkeley, 1996.
[MK00] Shintaro Meki and Yahiko Kambayashi. Acceleration of Relational Database Operations on Vector Processors. Systems and Computers in Japan, 31(8):79–88, 2000.
[OLG+ 07] John D Owens, David Luebke, Naga Govindaraju, Mark Harris,Jens Kruger, Aaron E Lefohn, and Timothy J Purcell. A survey of general-purpose computation on graphics hardware. In Computer graphics forum, volume 26, pages 80–113. Wiley Online Library, 2007.
[PJS97] Subbarao Palacharla, Norman P. Jouppi, and J. E. Smith. Complexity-Effective Superscalar Processors. In Proceedings of the 24th Annual International Symposium on Computer Architecture, pages 206–218, 1997.
[POR12] Polychroniou, Orestis, and Kenneth A. Ross. "High Throughput Heavy Hitter Aggregation." (2012).
[QCEV99] F. Quintana, J. Corbal, R. Espasa, and M. Valero, “Adding a Vector Unit to a Superscalar Processor,” in Proceedings of the 13th International Conference on Supercomputing, ser. ICS, 1999, pp. 1–10.
[RCJ11] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, “DRAMSim2: A Cycle Accurate Memory System Simulator,” IEEE Computer Architecure Letters, vol. 10, no. 1, pp. 16–19, Jan. 2011. ACM, 2008, pp. 18:1–18:15.
[Rus78] Richard M. Russell. The CRAY-1 Computer System. Communications of the ACM, 21(1):63–72, January 1978.
[Sch87] W. Schönauer. Scientific Computing on Vector Computers. Elsevier Science Publisher B.V., 1987.
[SCS+ 08] Larry Seiler, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, Jeremy Sugerman, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan, and Pat Hanrahan. Larrabee: A Many-core x86 Architecture for Visual Computing. In ACM SIGGRAPH 2008 Papers, SIGGRAPH ’08, pages 18:1–18:15. ACM, 2008.
[SFS00] J. E. Smith, Greg Faanes, and Rabin Sugumar. Vector Instruction Set Support for Conditional Operations. In Proceedings of the 27th Annual International Symposium on Computer Architecture, pages 260–269, 2000.
[Sut05] Herb Sutter. The free lunch is over: A fundamental turn toward concurrency in software. Dr. Dobbs Journal, 30(3):202–210, 2005.
[Tra11] Transaction Processing Performance Council. TPC-H Standard Specification v2.14.2. http://www.tpc.org/tpch/, 2011.
[WPB+ 09]Thomas Willhalm, Nicolae Popovici, Yazan Boshmaf, Hasso Plattner, Alexander Zeier, and Jan Schaffner. SIMD-Scan: Ultra Fast in-Memory Table Scan using on-Chip Vector Processing Units. Proceedings of The VLDB Endowment, 2(1):385–394, 2009.
[Your07] M. Yourst, “PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator,” in IEEE International Symposium on Performance Analysis of Systems Software, ser. ISPASS, 2007, pp. 23–34.
[ZR] Jingren Zhou and Kenneth A. Ross. Implementing database operations using simd instructions. In SIGMOD ’02, pages 145–156.