Accelerators for Technical Computing: Is It Worth the Pain? A TCO Perspective

Sandra Wienke, Dieter an Mey, and Matthias S. Müller

Center for Computing and Communication, RWTH Aachen University, D-52074 Aachen
JARA – High-Performance Computing, Schinkelstr. 2, D-52062 Aachen
{wienke,anmey,mueller}@rz.rwth-aachen.de

Abstract. Nowadays, HPC systems emerge in a great variety including commodity processors with attached accelerators which promise to improve the performance per watt ratio. These heterogeneous architectures often get far more complex to employ. Therefore, a hardware purchase decision should not only take capital expenses and operational costs such as power consumption into account, but also manpower. In this work, we take a look at the total cost of ownership (TCO) that includes costs for administration and programming effort. From that, we compute the costs per program run which can be used as a comparison metric for a purchase decision. In a case study, we evaluate our approach on two real-world simulation applications on Intel Xeon architectures, NVIDIA GPUs and Intel Xeon Phis by using different programming models: OpenCL, OpenACC, OpenMP and Intel's Language Extensions for Offload.

Keywords: TCO, heterogeneous architectures, GPU, Intel Xeon Phi, programming effort, OpenCL, OpenACC, OpenMP, Intel LEO, energy efficiency.

1 Introduction

On the way to exascale computing, the HPC community is aiming at increasing system performance while keeping a tight rein on its power consumption. To this end, performance per watt has become a common metric to compare different hardware architectures and nowadays heterogeneous HPC systems – combining commodity processors with accelerators – are in front of this efficiency comparison [1].

However, relying solely on the (HPL) performance per watt evaluation and a low energy bill is not advisable for hardware purchase decision making in university computing centers. Especially, their needs concerning the applications should signify for the assessment since they influence power consumption and performance. In this case, comparing comprehensively total costs of ownership (TCO) of different systems is an appropriate approach. While TCO calculations comprise numerous parameters such as acquisition and operational costs, the manpower costs for programmers are often not taken into account. The programming effort that is needed to port a code to and fully exploit the desirable hardware may significantly differ depending on the complexity of a (heterogeneous) architecture and of its programming model.

In this paper, we make a first approach to quantify these total costs including the programming effort for different hardware architectures and programming models. Thereby,

J.M. Kunkel, T. Ludwig, and H.W. Meuer (Eds.): ISC 2013, LNCS 7905, pp. 330–342, 2013.

we want to investigate the question whether it is preferable to buy accelerators. Assuming a given fixed one-time budget and a homogeneous application scenario, we compute the cost per program execution for several hardware architectures and use it as a comparison metric. We illustrate our approach in a case study based on our experience at RWTH Aachen University. We look at Intel Sandy Bridge servers and compare them to nodes with an attached NVIDIA Fermi GPU or Intel Phi coprocessor. Our case study considers two real-world simulation codes from the fields of engineering and biomedicine that serve as a basis for values in programming effort, performance and power consumption. We show results for the following programming approaches: OpenMP on Intel servers, a combination of OpenMP and Intel's Language Extensions for Offload (LEO) on Intel's Xeon Phi and OpenCL and OpenACC on NVIDIA's Fermi GPU.

The paper is structured as follows: Sect. 2 covers related work. In Sect. 3, we explain our TCO perspective and introduce the cost per program run as comparison metrics between system types. In the case study in Sect. 4, we investigate whether the accelerators effort pays off. Finally, we summarize our findings in Sect. 5.

2 Related Work

In data center total ownership costs, different metrics were established. Carlyle [2] looks at node hour cost for comparison of community centers with cloud computing services, while Turner and Seader [3] propose a combination of cost per square foot and cost per watt due to recent power and cooling considerations. Applying cost per watt, Patterson et al [4] compare different data center density designs while only giving some major cost deltas. In contrast, we compute absolute total ownership costs including comprehensive capital and operational expenses (called "True TCO" by Koomey [5]) and further compare different system types by the metric "cost per program run" given a fixed investment and system lifetime. Basing on our attempt to estimate the TCO of the RWTH Aachen University's HPC equipment in [6], we additionally quantify manpower costs for operating and programming different compute nodes. While administration effort is considered in few TCO calculations (e.g. [5]), programming effort is rarely investigated as a quantified TCO aspect for computing centers. Kuck [7] elaborates on productivity issues in HPC that also include development effort and defines TCO as the sum of total cost of purchase (TCP), total cost of operation and maintenance (TCOM) and total cost of applications development (TCAD). However, his approach misses to quantify TCO in real numbers. Simultaneously, software estimation models are aware of development and maintenance effort while not putting hardware costs into the equation [8]. The emergence of accelerators in high-performance technical computing increases the TCO complexity and makes a fair comparison to nodes with commodity processors challenging. In previous works [9,10], we have seen the importance of development productivity especially on (GPU) accelerators, also dependent on the used programming model and the kind of application. CAPS' case study pamphlet [11] gives a short overview of the economics of GPU code migration and draws the conclusion that GPUs are worthwhile when gathering at least a two-fold speedup. Besides a carefully investigated TCO calculation of Intel servers and an NVIDIA Fermi GPU, we will also take a look at the just recently released Intel Xeon Phi.

3 Total Cost of Ownership

"Total cost of ownership represents the cost to the owner to purchase/build, operate and maintain a data center" [12] and comprises numerous parameters such as capital expenses, energy and maintenance costs. Defining and quantifying all parameters is a challenging task that we have just started to tackle with respect to the RWTH's high-performance computing center. Our focus is to include administration and especially programming efforts for novel heterogeneous architectures into the TCO calculation. In the following, we divide TCO in one-time and annual costs and differ between per-node and per-node-type costs that arise for each compute device or just once for each system type, respectively. Table 2 gives an overview of all components. For modeling these TCO parameters, a spreadsheet is publicly available on our webpage [13].

3.1 One-Time Costs

One-time costs comprise the initial expenses for hardware, building and infrastructure, but also for manpower. Concerning hardware costs, we look at computing facilities, whereby costs for storage and networking can be included into the infrastructure component. Manpower costs arise for the installation of an operation system (OS), the environment setup and for developing or porting user applications that leverage the investigated architecture kind. They may differ highly depending on the system type, ease of use of the programming language, availability of tools and the base knowledge of the employees. The latter makes it also challenging for fair comparisons and is an issue that should be further addressed in future. All together, we define the one-time costs C_{ot} by

$$C_{ot} = C_A \cdot n + C_B$$

where C_A is the sum of all one-time costs per node, C_B the sum of all one-time costs per node type and n is the number of nodes that can be bought by a given investment I.

3.2 Annual Costs

Annual costs aggregate expenses for maintaining the hardware, the OS, the software environment and the user applications, but energy costs got the most attention in recent exascale discussions. To this end, we have to keep in mind that power consumption rises with performance and that in an accelerator machine the host processor also uses energy. The costs per anno C_{pa} are given with C_C the sum over all annual costs per node, C_D the sum over all annual costs per node type and n the number of nodes:

$$C_{pa} = C_C \cdot n + C_D$$

Combining both cost types, we define the total cost of ownership as a function of the number of nodes n and the lifetime τ of the system:

$$\text{TCO}(n, \tau) = C_{ot} + C_{pa} \cdot \tau = (C_A + C_C \cdot \tau) \cdot n + C_B + C_D \cdot \tau \quad (1)$$

Given a fixed budget I and lifetime τ (which is between 3-6 years in most HPC centers), we can compute the number of nodes n by solving (2) for n.

$$\text{Investment } I = \text{TCO}(n, \tau) \quad (2)$$

3.3 Costs per Program Run

On the basis of the TCO (1) for each architecture type, we investigate a comparison metric that further takes the gained application performance or rather the parallel runtime into account. As a starting point, we assume that just one application runs all the time. Thus, we can compute the number of application executions for each system type n_{ex} and the costs per program run C_{ppr} for n nodes by:

$$C_{ppr}(n,\tau) = \frac{\text{TCO}(n,\tau)}{n_{ex}(\tau)\cdot n} \quad \text{with} \quad n_{ex}(\tau) = \frac{k\cdot\tau}{t_{par}}$$

Here, τ is the system's lifetime, t_{par} is the parallel runtime of the application and k denotes the system usage rate in percent. The latter is introduced to account for additional scheduling times, maintenance periods or unreliability of the system. The costs per program run C_{ppr} can serve as comparison metric for different architecture types.

One interesting aspect for comparison is the break-even point with respect to investment, i.e. what investment is needed so that one system type (and programming model) is beneficial over another (given a fixed lifetime). We can derive it by looking for zeros in the difference for costs per program run of two system types X and Y:

$$C_{pprX}(n_X,\tau) - C_{pprY}(n_Y,\tau) = 0 \tag{3}$$

Using (2), we can extract n as a function of I and substitute it into the equation above. Then, we get as break-even point I_{be}:

$$I_{be} = \frac{n_{exY}(C_{AX}+C_{CX}\tau)(C_{BY}+C_{DY}\tau) - n_{exX}(C_{AY}+C_{CY}\tau)(C_{BX}+C_{DX}\tau)}{n_{exY}(C_{AX}+C_{CX}\tau) - n_{exX}(C_{AY}+C_{CY}\tau)} \tag{4}$$

4 Case Study on Accelerators for Technical Computing

Today's hype for accelerators motivates us to evaluate their benefit (or "pain") compared to Intel servers from a TCO perspective.

4.1 Real-World Applications

For the integration of manpower efforts, performance and power consumption into the TCO calculation, we look at two different real-world simulation codes since results simply based on benchmarks like HPL can be misleading. We chose kernels from these software packages that are generally suitable for accelerators, while keeping in mind that many applications are not. The kernels are small enough so that we could implement different versions with acceptable effort.

Neuromagnetic Inverse Problem. The application *NINA* comes from the field of biomedicine, or more precisely, magnetoencephalography. The arising neuromagnetic inverse problem deals with the reconstruction of focal activity in the brain and can be

solved by means of a p-norm minimization. For this unconstrained nonlinear optimization problem, the software package of Bücker et al [14] employs first- and second-order derivatives with automatic differentiation. It is implemented primarily in MATLAB, whereas the objective function, its first- and second-order derivatives are written in C to enable parallel computing. We parallelized these three kernels that include the computations of matrix-vector products using a matrix of dimensions 128×512000. The kernels account for ~ 100 kernel code lines and 90 % of the whole application's runtime.

Simulation of Bevel Gear Cutting. The engineering application *KegelSpan* [15], written in Fortran and developed by the Laboratory for Machine Tools and Production Engineering (WZL) at RWTH Aachen University, is a 3D simulation software for the bevel gear cutting process and applied in the automotive industry. It aims at minimizing the number of expensive tool changes in the bevel gears manufacturing process by enabling a detailed tool load and wear analysis. For the optimization of manufacturing parameters, the intersection of tool and gear is computed repeatedly where each run iterates million to billion times. Although this part is the biggest hotspot in the KegelSpan package, it only accounts for approx. 25 % of the serial runtime. Since its industry costumers have had GPU hardware at their disposal anyway, we still started accelerating this kernel [9] (~ 150 lines in serial code) using the portable OpenCL. However, for our TCO calculations, we will assume that this module accounts for 90 % of the whole application runtime to illustrate our statements and not be restricted by Amdahl's law.

4.2 TCO Components

The following numbers and assumptions about the TCO components are based on our experience at the Center for Computing and Communication at RWTH Aachen University. Here, one possible scenario may belong to our integrative hosting activities that allow other RWTH members to integrate their HPC equipment into our cluster environment. We assume that an RWTH professor with a certain budget wants to buy compute nodes to accelerate his one research application (i.e. given an homogeneous application landscape). At the computing center, we want to give the professor an estimation which architecture is worth to be purchased by taking also programming effort into account.

System Types. We start by gathering results for single compute nodes and assume that results can be extrapolated to a cluster amount using (2). Furthermore, investigations that include network communication across nodes and the associated programming effort (e.g. MPI) are left for future work. The different system types (*ST*) are running Scientific Linux 6.3 and are given as follows:

ST1: As an X86 base, we take an Intel Sandy Bridge server which has widely been accepted as a cost-efficient architecture for compute services. It contains two-socket Intel Xeon E5-2650 CPUs running at 2.00 GHz with a total of 16 cores and 32 GB memory.

ST2: This accelerator architecture contains an NVIDIA Tesla C2050 (Fermi) GPU with ECC enabled. The host system consists of a 4-core Intel Westmere processor (Xeon

Table 1. Programming effort (in man-days) and kernel speedup w.r.t. the serial versions of NINA and KegelSpan. The power consumption (in watt) is taken from the whole system during the kernel execution.

		OMP-simp/ST1	OMP-vec/ST1	OCL/ST2	OpenACC/ST2	LEO/ST3
NINA	Effort	1	5	7	4	6
	Speedup	6.58	7.56	9.92	3.09	11.33
	Power	191.99	200.56	284.09	293.23	277.65
Kegel Span	Effort	0.5	3.5	5	1.5	4.5
	Speedup	15.47	22.55	46.65	47.02	44.16
	Power	166.17	155.03	249.64	227.36	230.87

E5620@2.40 GHz) with 12 GB of memory. Since we did not have access to an up-to-date NVIDIA Kepler GPU at time of writing, it will be subject of future investigations.

ST3: The second accelerator type comprises an Intel MIC coprocessor – an Intel Xeon Phi 5110P with 60 cores running at 1.053 GHz and 8 GB memory. For sound comparisons, we just assume that this machine has the same host configuration as the NVIDIA GPU system. We adapted accordingly hardware prices and power consumption, but took the real-measured kernel runtimes and speedups since the host processor does not significantly contribute to these results. In real life, the host system equals *ST1* and it contains two Intel Phis of the given type. As it is an early machine, not all settings (especially concerning energy) may be optimally configured yet.

Programming Effort. In previous works [9,10], we investigated the impact of development effort in accelerating code regions and expressed it by the number of lines of kernel code added or modified. Now, we try to quantify this effort in man-days (see Tab. 1) for inclusion into real-cost calculations and thereby put them into perspective on our way towards improving future purchase decisions. Therefore, we measured the development time of the first parallel version for both applications. It includes time for programming, debugging and analysis and therefore is also dependent on the available tools supporting the programming model. Efforts are based on a moderately-experienced programmer who already knows details on the hardware architecture and the programming paradigm. For the following implementations, we could directly apply some code lessons learned during the first implementation. Therefore, we added some approximated time to the real-measured one to be able to compute costs independently. The development days in Tab. 1 correspond to each best-effort version. We developed five parallel variants of each kernel. We started with a simple OpenMP version (*OMP-simp*) by applying OpenMP directives to the original serial code. The *OMP-vec* version includes code restructuring for (auto-)vectorization and further parallelization with OpenMP. Programming an accelerator puts much more restrictions on programmability. In the case of a GPU, the number of threads has to be very high to overcome latencies, and the brand-new Intel Xeon Phi only performs well if many threads execute highly-vectorized code. For GPUs, we tuned the application with OpenCL (*OCL*) by reducing

data transfers and using (if applicable) GPU on-chip memory, pinned memory, asynchronous execution and more. In the *OpenACC* version, we used directives to offload code regions to the GPU and tried to get the code as close as possible to the OpenCL version. For the Xeon Phi, we combined OpenMP with Intel's LEO while decreasing data transfers, applying data asynchronous movement and focusing on vectorization and vector alignment. Additional code changes must be applied to overcome small nearly-serial parts that were not an issue for tens of threads but became one for hundreds. Our GPU and Phi versions perform the compute-intensive kernels on the accelerator while the host does not contribute to performance gains. However, we will deal with estimations for full heterogeneous versions in Sect. 4.3. Going into depth in Tab. 1, we see that restructuring code for vectorization is time consuming. Especially for Xeon Phi, that is said to be easily programmable, the development effort rises as vectorization is really important to get performance. However, assuming that a highly-vectorized code version does already exist for the host, the denoted programming effort can be decreased. Furthermore, low-level GPU code development needs more manpower than parallelization on CPUs (*OMP-simp*, *OMP-vec*). Only directive-based GPU programming (with *OpenACC*) may decrease this effort.

Performance. The reported performance results include data transfers between host and device, kernel execution times and the overhead introduced by the need to adapt the data structure. The speedups shown refer to the kernel runtimes, but we use whole application speedups for the TCO calculation. For comparisons, the speedup values are given with respect to the serial version measured on *ST1*. However, in the following examinations, we compare appropriately performance to the *OMP-simp* version which uses all cores of the hardware. OpenMP (*ST1*) and Xeon Phi (*ST3*) results are gathered using the Intel compiler 13.0.1. We started 16 threads on *ST1* and 177 and 118 threads on Xeon Phi (*ST3*) for NINA and KegelSpan best-effort results, respectively. OpenCL (*ST2*) relies on CUDA toolkit 5.0 and OpenACC (*ST2*) on the PGI compiler 12.9 [1] that uses CUDA toolkit 4.1. Optimization flags are used as well (e.g. -O3 or fastmath). For both codes, the kernel speedups we can achieve on any accelerator are about 2-4x relative to the baseline *OMP-simp* implementation on *ST1*. While most performance results are as expected, we notice that NINA's OpenACC performance is rather disappointing. The reason probably lies in not yet fully-implemented OpenACC features in the PGI compiler (more details in [10]) which might be tackled in future compiler releases.

RWTH One-Time Costs. Diving into Tab. 2, the TCO components are listed based on the RWTH environment and with respect to the NINA application. Considering the hardware purchase, we use list prices for current generation servers and workstations, kindly provided by the company Bull in January 2013. For evaluations of TCO calculations with numbers from other sources, we provide the editable TCO spreadsheet [13]. In order to estimate the infrastructure costs for housing the compute devices, we express the actual one-time building costs as annual costs of 200,000€ [6] over 4 years of system lifetime. Breaking down this total annual cost per node, we divide it by a

[1] Recent PGI compiler versions (13.1 - 13.3) were not used due to a compiler problem that evokes a performance loss in our case.

Table 2. One-time and annual costs in € for the NINA application

		one-time costs C_{ot}					annual costs C_{pa}					
		per node			per node type		per node				per node type		
		HW purchase	Building/ infrastruct.	OS/env installation	OS/env. installation	Prog. effort	HW maintenance	Building/ infrastruct. per watt	OS/env maintenance	Energy	OS/env. maintenance	Compiler/ software	Application maintenance
ST1	OMP-simp	7,137	0	0	0	286	300	33	78	317	0	0	0
ST1	OMP-vec	7,137	0	0	0	1,429	300	33	78	322	0	0	0
ST2	OCL	7,713	0	0	0	2,000	324	49	78	421	0	0	0
ST2	OpenACC	7,713	0	0	0	1,143	324	49	78	506	0	0	0
ST3	LEO	9,644	0	0	0	1,714	405	49	78	465	0	0	0
		C_A			C_B		C_C				C_D		

total of 1.6 MW of energy consumption, which today is the limiting factor for housing machinery in this building [6], and multiply it by the maximum power consumption of each compute node. The initial administration effort for integrating and installing novel (accelerator) systems is high: The staff must first get to know the systems, establish operating concepts, integrate it into the existing batch scheduler, install new drivers and software and may make sure that further maintenance updates can be easily rolled out to all nodes of a system type simultaneously. If we set up a TCO calculation for a single system, such an effort would never pay off. On the other hand, it seems that university computing centers cannot avoid investigating the usability of recent accelerator architectures due to increasing power bills. Since our administration staff has already gained experiences in the past, we assume no extra one-time charges for new installations in our calculations. While transferring the programming effort that we explained previously into manpower costs, we assume the cost of one day of a full time employee (FTE) at 285.71 € in accordance to funding guidelines of the German Science Foundation [16] and the the European Commission's CORDIS [17].

RWTH Annual Costs. The annual costs per node include hardware maintenance that is provided by the vendor (Bull) and accounts for 5 % of the purchasing costs. At RWTH Aachen University, 4 administrative FTEs are running the whole compute cluster [6] and 75 % of the manpower accounts for annual maintenance. We quantify the administration effort per node by dividing the 180,000 € manpower costs [16] by the total number of nodes in our cluster (roughly 2,300) and get approximately 78 € per any kind of compute node. There is no significant additional effort per node type since a generic approach to roll out software was established during the first installation. Furthermore, we do not have any additional software or compiler costs as we buy the licenses anyway for the whole cluster (e.g. Intel or PGI compiler) or the software is free of charge (e.g. CUDA toolkit). Energy costs are dependent on the hardware, the running application, the power usage effectiveness (PUE) of the computing center and

Table 3. Costs per program run C_{ppr} given an investment of 250,000 € and a lifetime of 4 years & break-even points I_{be} in investment (see (4)) w.r.t. OMP-simp for NINA and KegelSpan.

		OMP-simp	OMP-vec	OCL	OpenACC	LEO
NINA	#nodes	24.84	24.69	22.14	21.57	18.21
	t_{par}[s]	106.87	98.92	86.10	176.72	81.01
	n_{ex}	944,265	1,020,119	1,172,090	571,034	1,245,711
	C_{ppr}[€]	0.01066	0.00993	0.00963	0.02030	0.01102
	I_{be}[€]		15,989	17,058	-	-
KegelSpan	#nodes	25.46	25.53	22.96	23.12	18.61
	t_{par}[s]	158.42	140.13	119.48	119.33	120.57
	n_{ex}	637,027	720,150	844,632	845,708	836,993
	C_{ppr}[€]	0.01542	0.01360	0.01289	0.01279	0.01605
	I_{be}[€]		7,231	7,787	1,809	-

the regional electricity cost. Here, we pay roughly 0.15 €/kWh and have an estimated PUE of 1.5 [6]. The power usage measurements were done using a Raritan Dominion PX power distribution unit on *ST1* and *ST2*. Since *ST3* is an artificial construction, we recorded the power consumption of the Xeon Phi during the kernel run and added the appropriate host consumption from *ST2*. In general, we further differ between the power consumption of the application kernel (compare Tab. 1) and the rest of the application that mostly runs sequentially. Finally, we set the costs for application maintenance to 0 € since the investigated kernels are quite small. However, one should keep in mind that this effort may increase especially for bigger codes when the chosen programming paradigm is verbose or when a lot of code restructuring is needed.

Doing the math, we compute the number of nodes that can be bought with a sample investment of 250,000 €, the number of executions n_{ex} (assuming the application runs 24/7 with a cluster usage rate of 80 %) and finally the costs per program run (Tab. 3).

4.3 Results

Based on our previous calculations, we interpret the results and perform a what-if analysis with focus on programming effort.

For both codes, NINA and KegelSpan, a simple OpenMP parallelization did not cost a lot of effort and thus increased the TCO only slightly while speeding up the compute-intensive kernels considerably. In order to investigate the cost/performance ratio of the accelerators, we take this simple OpenMP version as the baseline. Figures 1 and 2 present graphs over the varied amount of budget which show the difference in cost per program run relative to the *OMP-simp* version in percentages. A negative percentage means that the given system type is $x\,\%$ cheaper than the *OMP-simp* version and vice versa. We added results for estimated hybrid OpenCL (*OCL-hyb*) and Xeon Phi (*LEO-hyb*) solutions. The numbers on the right hand side of the figures express the limiting values for infinitely-big investments.

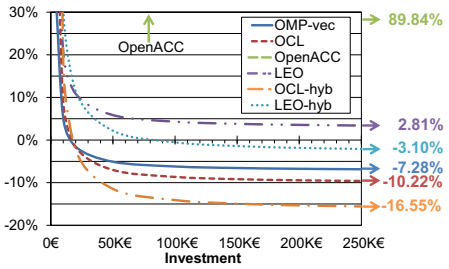

Fig. 1. NINA's cost per program run in percent relative to OMP-simp

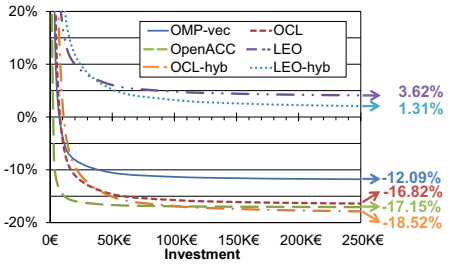

Fig. 2. KegelSpan's cost per program run in percent relative to OMP-simp

NINA. For the NINA software package (Fig. 1), the *OMP-vec* version is already worthwhile given an investment of 15,989€ or more (see Tab. 3). If more budget is available (at least 17,000€), GPU accelerators and OpenCL become profitable. For the estimated OpenCL-hybrid solution, the initial programming effort would also pay off. On the other hand, the low programming effort of the *OpenACC* version does not make up for its slowness and we would pay roughly 90 % more than the *OMP-simp* version costs per program run with an infinite investment. Looking at Intel's Xeon Phi accelerator, it is surprising that it cannot beat the simple OpenMP version when taking the total ownership costs into account (the limit amounts to $\sim 3\,\%$). However, compared to the OpenCL version, its programming effort is lower and the performance a bit better. The power consumptions plays a role, but the higher hardware purchase costs have the main impact on the TCO. For the new NVIDIA Kepler GPU, we would probably see a similar picture. The estimated *LEO-hyb* version would become at least advantageous over the *OMP-simp* version given a budget of 83,000€ or more.

KegelSpan. For KegelSpan, Fig. 2 illustrates a profitable *OMP-vec* version like for NINA. In contrast, the *OpenACC* accelerator version is already beneficial over the OpenMP simple host version given an investment of 1,800€ (see Tab. 3). This is due to low programming effort and the good performance. The other GPU accelerator versions are profitable as well while our estimated *OCL-hyb* is only slightly more beneficial than the *OpenACC* and the *OCL* version. In contrast, the Xeon Phi accelerator does not pay off. Even our hybrid estimation would run into a positive percentage of 1.3 for infinitely-big investment.

What-If Analysis. Taking the data discussed as foundation, we perform a "what-if" analysis by varying certain components of the TCO calculation.

First, we look at the impact of Amdahl's law on the KegelSpan application. We plot the break-even points in investment I_{be} (compare (4)) as a function of the kernel portion in Fig. 3. It illustrates that bigger kernel portions – and thereby higher performance – results in lower investment needed to compensate the higher initial expenses over the *OMP-simp* version. The *OMP-vec* version pays off for small kernel portions from a moderate investment on. On the contrary, the GPU variants need a kernel portion around 75 % to be beneficial at all. Then, the *OpenACC* version can require a lower budget than

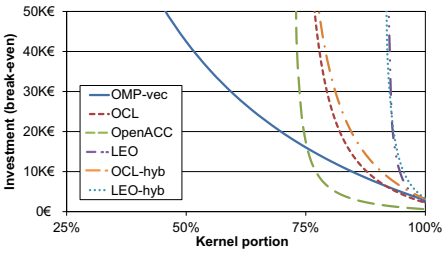

Fig. 3. Break-even points in investment over OMP-simp as function of KegelSpan's kernel portion

the *OMP-vec* version to be advantageous, whereas the OpenCL versions are roughly at the same level for big kernel portions. The Phi coprocessor variants even need 92 % kernel portions to be beneficial, i.e. the performance pays off the hardware costs. To this end, we see that the comparison of system types and programming models is also effected by Amdahl's law and that our results cannot be taken as absolute statements.

Both of our investigated kernels were moderately small. The question with bigger codes is how the higher programming effort influences the results. Given the TCO and break-even formulas, we can derive that the increase of kernel code means a proportional increase of the needed minimal investment to make a program run beneficial.

The introduced hybrid estimations assumed a certain programming effort given the optimal performance based on vectorized host and accelerator speedups. Now, we turn round the perspective and investigate whether and when a hybrid implementation would be worthwhile. Taking (3), we solve it for the development effort and thereby get the desired break-even point in man-days. We anticipate that a hybrid OpenCL implementation may need up to 146 and 162 man-days, respectively for NINA and KegelSpan, to still be beneficial over the *OMP-simp* version. In contrast, development may only take up to 28 man-days for a heterogeneous NINA Phi solution, but a hybrid Phi implementation for KegelSpan would never pay off. The latter may be canceled out by a more efficient host system.

5 Conclusion

In the context of one case study at RWTH Aachen University, we examined the benefit of accelerators in technical computing. Based on total ownership costs (TCO), we compared costs per program run for specific applications on an Intel server, NVIDIA Fermi GPU and Intel Xeon Phi coprocessor while putting human effort into the equation.

Taking a simple OpenMP version as baseline, we find that most GPU Fermi solutions pay off additional manpower efforts. Furthermore, OpenACC GPU results illustrated that the ratio of performance per development time and power consumption is interesting: Low programming effort, but low performance is expensive (see NINA application), whereas a combination with good performance rocks (compare KegelSpan results). On the other hand, results gathered on Intel's Xeon Phi were surprisingly disappointing. Here, the system acquisition costs were mainly responsible for the non cost-efficient result. Additionally, it took quite some effort to create solutions with good

performance due to vectorization tuning, despite that the Xeon Phi is said to be easily programmable. However, if highly-vectorized host implementations are available as baseline, this picture will improve (a déjà vu for elder vector computer users). Generally, host systems are usually less expensive, require less power and need less manpower so that the accelerators' performance has to compensate these three aspects. A perspective based just on performance per watt is limited.

Furthermore, our TCO model (available at [13]) allows projecting the feasibility of considered solutions such as hybrid implementations. For instance, our results show that the according effort does not always pay off (depending on hardware and performance).

In future, we will include additional programming paradigms like the upcoming OpenMP 4.0 features and new architectures like NVIDIA's Kepler GPU. We will further take network communication (MPI) into account to balance cost efficiency versus real-time constraints. Since we have assumed a homogeneous application landscape so far, future examinations will look at the impact of mixed job executions. We also want to turn our TCO analysis of experimentally-gathered data into an analytical model with predictive powers. Performance models that can predict the parallel runtime from the single-core code performance do already exist. Given the recent interest in reducing machine power envelopes, the existing power consumption models will hopefully get enhanced in the near future. Additionally, a reliable model to estimate the manpower efforts is also essential in order to improve the quality of the TCO interpretations. We started measuring the amount of effort that our students put into program development during their practical trainings in order to obtain baseline data as a first step to modeling programming productivity.

References

1. The Green 500: The Green500 List - November 2012: Heterogeneous Systems Re-Claim Green500 List Dominance (2012), `http://www.green500.org/lists/green201211`
2. Carlyle, A., Harrell, S., Smith, P.: Cost-Effective HPC: The Community or the Cloud? In: CloudCom 2010, pp. 169–176 (2010)
3. Turner, W.P., Seader, J.H.: Dollars per kw plus dollars per square foot are a better data center cost model than dollars per square foot alone. Technical report, Uptime Institute (2006)
4. Patterson, M., Costello, D., Grimm, P., Loeffler, M.: Data center TCO; a comparison of high-density and low-density spaces. Technical report, Intel Corporation (2007)
5. Koomey, J.: A Simple Model for Determining True Total Cost of Ownership for Data Centers. Technical report, Uptime Institute (2008)
6. Bischof, C., Mey, D.a., Iwainsky, C.: Brainware for green HPC. Computer Science - Research and Development 27, 227–233 (2012)
7. Kuck, D.J.: Productivity in high performance computing. Int. J. High Perform. Comput. Appl. 18(4), 489–504 (2004)
8. Galorath, D.D.: Software Total Ownership Costs: Development Is Only Job One. Software Tech. News 11(3) (2008)
9. Wienke, S., Plotnikov, D., Mey, D.a., Bischof, C., Hardjosuwito, A., Gorgels, C., Brecher, C.: Simulation of Bevel Gear Cutting with GPGPUs – Performance and Productivity. Computer Science - Research and Development 26, 165–174 (2011)

10. Wienke, S., Springer, P., Terboven, C., an Mey, D.: OpenACC — First Experiences with Real-World Applications. In: Kaklamanis, C., Papatheodorou, T., Spirakis, P.G. (eds.) Euro-Par 2012. LNCS, vol. 7484, pp. 859–870. Springer, Heidelberg (2012)
11. CAPS enterprise: Case study – Code Migration to CPU-GPU Hybrid: An Economic Approach (2010)
12. Wang, L., Khan, S.: Review of performance metrics for green data centers: a taxonomy study. The Journal of Supercomputing, 1–18 (2011)
13. Wienke, S., an Mey, D., Müller, M.S.: Accelerators for Technical Computing: Is it Worth the Pain? TCO Spreadsheet (2013), `https://sharepoint.campus.rwth-aachen.de/units/rz/HPC/public/Shared%20Documents/WienkeEtAl_Accelerators-TCO-Perspective.xlsx`
14. Bücker, H.M., Beucker, R., Rupp, A.: Parallel Minimum p-Norm Solution of the Neuromagnetic Inverse Problem for Realistic Signals Using Exact Hessian-Vector Products. SIAM Journal on Scientific Computing 30(6), 2905–2921 (2008)
15. Brecher, C., Gorgels, C., Hardjosuwito, A.: Simulation based Tool Wear Analysis in Bevel Gear Cutting. In: International Conference on Gears, Düsseldorf. VDI-Berichte, vol. 2108.2, pp. 1381–1384. VDI Verlag (2010)
16. German Science Foundation (DFG): Personalmittelsätze der DFG für das Jahr 2013 (2013)
17. Community Research and Development Information Service: Audit Certificate Guidance Notes 6th Framework Programme (2005)