Modeling CPU Energy Consumption of HPC Applications on the IBM POWER7

Philipp Gschwandtner*†, Michael Knobloch*, Bernd Mohr*, Dirk Pleiter* and Thomas Fahringer†
*Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, 52425 Jülich, Germany
†University of Innsbruck, Technikerstrasse 21a, 6020 Innsbruck, Austria

***Abstract*—Energy consumption optimization of HPC applications inherently requires measurements for reference and comparison. However, most of today's systems lack the necessary hardware support for power or energy measurements. Furthermore, in-band data availability is preferred for specific optimization techniques such as auto-tuning. For this reason, we present in-band energy consumption models for the IBM POWER7 processor based on hardware counters. We demonstrate that linear regression is a suitable means for modeling energy consumption, and we rely on already available, high-level benchmarks for training instead of self-written or hand-tuned micro-kernels. We compare modeling efforts for different instruction mixes caused by two compilers (GCC and IBM XL) as well as various multi-threading usage scenarios, and validate across our training benchmarks and two real-world applications. Results show mean errors of approximately 1% and overall max errors of 5.3% for GCC.**

I. Introduction

Although high computational performance is still the main objective in most fields concerned with parallel computing, there have been research efforts towards power and recently also energy consumption minimization. The reasons for that are diverse, ranging from infrastructural limits such as the 20 MW power limit for exascale systems [1] or cooling equipment requirements over financial constraints to environmental considerations. While some of these aspects originally target power consumption, there are new aspirations directly concerned with energy.

However, as of today it still remains unclear how to meet all these requirements. While industry implemented numerous technologies in hardware over the years, such as DVFS, clock and power gating, et cetera, deployed systems still provide only limited support for measuring power and energy consumption. Nevertheless, optimizing for power or energy inherently requires measurements for reference and comparison, and depending on the specific research problem only few suitable solutions exist.

In the overall power and energy consumption of an HPC node without accelerators, the CPU is currently the biggest contributor that can also be influenced the most depending on the system configuration and the workload executed. As such, most efforts concentrate on this part of a system and this work also sets its focus there. There are several instrumentation systems available [2], [3] that supply researchers with power and energy consumption information. However, many suffer from topological granularity issues as well as resolution and accuracy problems. In addition, many of them supply results out-of-band, impeding or preventing many new optimization efforts in HPC such as auto-tuning [4] or performance analysis [5] with respect to energy.

There has been a lot of research invested in the modeling of power or energy consumption, either via architectural simulators or statistical models coupled with reference measurements. In this work we take the latter path and present energy models for an IBM POWER7 processor.

Our main contributions are simple yet accurate energy models for the POWER7 that – once calibrated – solely rely on information easily obtainable in-band by most performance analysis tools and frameworks. Specifically, we use hardware counters and correlate their data with the energy consumption of the chip using linear regression. We do this with respect to various benchmarks and applications as well as different parallelism and compiler setups that cause different instruction mixes. To the best of our knowledge, there has been no other energy modeling work that compares compiler effects and parallelism. We achieve high accuracy with a maximum error of 5.3% and an average error of ~1% when using GCC. Furthermore, we show that – contrary to popular approaches in related work – it is not always necessary to train models with micro-benchmarks that are specifically tuned to the hardware in question. This shows that such models can be easily portable.

The paper is organized as follows: Section II outlines related work. Section III will present a brief description of the hardware characteristics of the POWER7 chip, focusing on aspects related to this research. Section IV will describe our methodology, define our setup and usage scenarios and list general conditions. The models themselves are discussed in Section V and the results presented and illustrated in Section VI. Finally Section VII concludes and provides an outlook for future work.

II. Related Work

There is an increasing amount of related work that deals with modeling power and energy consumption. Kestor et al. [6] developed a per-core power model for an AMD Interlagos system and suggest the use of a System Management Interface (SMI) to enable easy access to power sensor values, whether they are measured or modeled. Similarly, Goel et al. built models [7] for different Intel and AMD platforms. Both approaches model power consumption whereas we investigate energy consumption modeling. Furthermore, both rely on coarse-grained node-level instrumentation while we use fine-grained CPU component power measurements instead. Similarly, a power model for an Intel Core i7 is presented

in [8], however it is based on a less accurate instrumentation infrastructure [2] and does not state maximum errors. To the best of our knowledge, no related work explored the effect of using different compilers.

In [9] Huang et al. present a per-core power proxy for the POWER7. They achieve a comparable mean error of 1.8% for their workloads, however their model relies on activity sensors that are not available in production (neither in- nor out-of-band) and they train with over 700 micro-benchmarks. A similar but production-deployed model exists for Intel Sandy/Ivy Bridge processors and others, and is used by their Running Average Power Limit (RAPL) interface [10]. It offers in-band energy estimations of the entire CPU, and depending on the CPU model also captures off-core entities such as the memory controller or the integrated graphics unit. Another, less documented in-hardware model called Application Power Management (APM) is in use on newer AMD processors [11].

There is also research that uses POWER7 measurements but does not attempt to model power or energy consumption. Instead they are used solely for analysis and optimization [12], [13] or power management purposes [14].

Finally there is a long history of architectural simulators that also give power or energy information [15], [16], [17]. They face the difficulty of simulating increasingly complex architectures, the availability of input parameters for specific CPU models and need to balance the trade-off between accuracy and simulation overhead. Approaches such as ours have the advantage of comparatively negligible overhead and no need for detailed processor specifications.

III. Architecture

The hardware platform for our experiments is an IBM Power 720 Express node equipped with a single POWER7 processor fabricated in 45 nm [18]. It features 4 out-of-order cores clocked at 3.0 GHz, with each core offering up to 4 hardware threads and a total of 12 execution units (among which 2 fixed-point units (FXUs), 4 double-precision (DP) floating point units, a vector unit and 2 load/store units (LSUs) capable of performing simple fixed-point operations). The floating point and vector units are combined into a single vector-scalar unit that can process 2 DP floating point or 4 integer operations simultaneously. Furthermore, each core holds 32 KB of L1 data and instruction cache each, as well as 256 KB of L2 cache. The entire CPU holds 16 MB of L3 cache, divided into 4 MB parts local to each core. The system is equipped with 16 GB of DDR3 main memory, accessible via two on-chip memory controllers. The operating system in use is openSUSE 12.2 with a 3.4.6 Linux kernel. Our compilers consist of GCC 4.7.1 and IBM XL compilers technology preview version 13.0 for C and 15.0 for Fortran.

In addition to its conventional architectural features, the POWER7 node is also equipped with power instrumentation via its Thermal and Power Management Device (TPMD). It includes several power, temperature and other sensors that monitor the entire node as well as its subsystems such as the CPU, memory or I/O. These sensors can be accessed via the POWER7's service processor and collected out-of-band using IBM's Amester tool, causing no performance or power/energy perturbation on the POWER7 itself. For the scope of this research, we confine our measurements to the CPU power, eliminating any noise originating from other components. The specific sensors we use are sampled at 1 kHz with a resolution of 100 mW and an accuracy that is no worse than 2% by design [19], [14]. The readings are then filtered by constructing averages every 32 ms. We then use this data and the total run time of our training benchmarks to derive the total energy consumed. The high resolution and documented accuracy of the sensors coupled with the high sampling frequency ensures high accuracy of the reference data.

TABLE I. Investigated compiler and parallelization setups. #P denotes the number of processes while #T denotes the number of threads per process. The number suffix in the setup name states the overall number of threads, the suffix _NOSMT denotes the cases run in SMT1 mode.

setup name	NAS		MP2C		SHS	
	#P	#T	#P	#T	#P	#T
GCC4_NOSMT	1	4	4	1	2	2
GCC4	1	4	4	1	2	2
GCC8	1	8	8	1	2	4
GCC16	1	16	16	1	2	8
XLC4_NOSMT	1	4	4	1	2	2
XLC4	1	4	4	1	2	2
XLC8	1	8	8	1	2	4
XLC16	1	16	16	1	2	8

IV. Methodology

Although using micro-benchmarks is a popular method in related work to train power and energy models [6], it often requires tuning such benchmarks to specific hardware characteristics. Considering the complexity of today's processors, including multi-threading and available execution paths, this introduces additional effort and possible causes for inaccuracy. To eliminate these, we rely on already available, higher-level benchmarks to stress the system and train our models. To that end, we choose the SNU C/OpenMP variants of version 3.3 of the NAS Parallel Benchmarks [20]. To arrive at a reasonable number of training codes, we arbitrarily select 15 benchmark-problem size combinations with a run time of approximately 500 ms to 90 s (see Table III for a list of the benchmarks and problem classes in use). To test the accuracy of our models we employ k-fold cross-validation with $k = 15$, meaning we train with 14 of these codes and use the 15th for validation. This is done 15 times, choosing each code for validation once in a rotating manner. All training benchmarks are run with the number of OpenMP threads according Table I.

We want to eliminate any false sense of accuracy by not restricting our validation codes to a closed set of same-source C/OpenMP codes. Therefore, we also include two real-world MPI applications in the validation process, MP2C [21] and SHS [22]. MP2C (Massively Parallel Multi-Particle Collision) simulates fluids with solvated particles, using both molecular dynamics as well as multi-particle collision dynamics. It is written in Fortran90 and exclusively uses MPI. SHS (Simple Hyperbolic Solver) solves the compressible Navier-Stokes equation, describing the motion of fluids, on a two-dimensional domain using a simple one-step explicit finite difference scheme. It is written in C and uses both MPI and OpenMP for parallelization.

We investigate energy modeling when using different compilers (GCC and XLC, with optimization levels -O3 and -O5

respectively). Moreover, we test various simultaneous multi-threading (SMT) usage scenarios since the POWER7 offers the capability to set the number of active hardware threads per core (modes SMT1, SMT2 and SMT4, with the digit denoting the number of enabled hardware threads). Specifically, we use the SMT4 mode (using 1, 2, and 4 threads per core, with any remaining hardware threads active but idle) and hardware SMT1 mode (using 1 thread per core) to examine their effect on energy consumption modeling. Table I lists all our usage scenarios with their respective names and the corresponding parallelization setups of our codes. Our affinity policy is scatter-first, meaning each process is mapped onto a dedicated core before sharing one with another process. Afterwards any remaining OpenMP threads are scheduled in the same manner, balancing the usage of hardware threads over all cores.

To minimize any inaccuracy that arises from CPU load noise due to the operating system and background processes, as well as any possible inaccuracies caused by the measurement system, all data reported in this work is the result of 10 runs (for both training and validation). From these 10 runs, we choose the median of each metric (hardware counter events, measured energy consumption, time).

Ordinarily, the processor temperature would need to be included in the input parameters of a power or energy model, since leakage power increases with the temperature [9]. This would pose a problem, since we found no well-documented in-band temperature sensor that is accessible to us. However, out-of-band measurements with Amester showed the temperature of our specific sample to remain between 28° C and 34° C when the processor was idle or under full load for longer periods. Upon further investigation of the power readings, we found the effect of this temperature variation to border on the power measurement resolution and the introduced error to be less than 0.5%.

V. Model

Our energy modeling efforts aim at several goals that we are trying to unite. The models should be

1) high-level,
2) in-band,
3) low in computational intensiveness in training and deployment, and
4) accurate enough for detailed energy consumption analysis and optimization of parallel programs.

To satisfy constraints 1) and 2), we only use hardware counter information that can be obtained via popular high-level interfaces such as PAPI [23]. Since 3) requires the models to not only to be computationally fast in deployment but also in training, we focus our method on ordinary least squares (OLS) multiple linear regression as opposed to more complex solutions such as artificial neural networks. The base idea is that on average each hardware event contributes to the overall energy consumption with a fixed quota. As we will show in Section VI, this is a simple yet sufficient approach for most cases to achieve goal 4), high accuracy.

Since we always consider full usage of the chip and only vary the number of active hardware threads per core, we model energy as

$$E_{\text{total}} = E_{\text{idle}} + E_{\text{dynamic}} \tag{1}$$

TABLE II. Hardware event types initially considered for modeling.

event name	event description
PAPI_FP_INS	Floating Point Operation Finished
PAPI_INT_INS	Fixed point unit 0 or 1 finished an instruction
PAPI_L1_DCM	L1 data cache misses
PAPI_L2_DCM	L2 data cache misses
PAPI_L3_DCM	L3 data cache misses
PAPI_L3_DCR	L3 data cache reads
PAPI_TOT_CYC	Total cycles
PAPI_TOT_INS	Total instructions
PM_CMPLU_STALL	CPU stalls due to any reason
PM_CMPLU_STALL_THRD	CPU stalls due to thread conflict
PM_L1_ICACHE_MISS	L1 instruction cache miss
PM_L2_INST_MISS	L2 instruction cache miss
PM_L3_MISS	L3 references that miss the L3 cache
PM_L3_PREF_MISS	L3 prefetches that miss the L3 cache
PM_LSU_DC_PREF _STREAM_CONFIRM	An active prefetching stream matches a load from a load/store unit
PM_LSU_FX_FIN	Load/store unit finished a fixed-point instruction
PM_LSU_LDF	Load/store unit finished a scalar load
PM_LSU_LDX	Load/store unit finished a vector load
PM_MEM0_PREFETCH _DISP	Prefetch memory read issued on memory controller 0
PM_VSU_FMA_DOUBLE	Vectorized fused multiply-add finished
PM_VSU_SIMPLE_ISSUED	Vector unit finished a simple VMX instruction
PM_VSU_VECTOR _DOUBLE_ISSUED	Vector unit finished a double-precision vector instruction
PM_VSU_VECTOR _SINGLE_ISSUED	Vector unit finished a single-precision vector instruction

$$E_{\text{dynamic}} = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_j c_{i,j} \tag{2}$$

where E_{total} denotes the overall energy consumption during the execution of a workload, E_{idle} and E_{dynamic} denote its idle and workload execution-dependent parts. To arrive at a value for E_{dynamic} our models sum all counter values $c_{i,j}$ of each event type $1 < i \leq n$ over all hardware threads $1 < j \leq m$ of all cores and multiplies them with their respective regression coefficients α_i. Then we compute the final result by summing over all event types.

Since the instrumentation system of the POWER7 does not directly offer energy but only power readings, we derive the idle and dynamic energy by including the execution time when performing our measurements. Nevertheless it should be noted that the time is *not* an input of the linear regression part of our models, but only used to derive energy values from power readings and to compute E_{idle}. This is also the reason why we do not include any idle cycles counter for the linear regression part, since it only models E_{dynamic}.

As our models are based on hardware counters, a main concern is their selection. Since the POWER7 offers over 500 different hardware counter events [24], a smaller subset must be derived. One approach in related work is to first manually select a medium number of hardware counters that seems like a good fit with regard to the hardware architecture [6]. We also employ this method of initial counter selection. However, in [6] a series of benchmarks is run afterwards and the number of counters further reduced by computing correlation coefficients between the counters themselves, as well as correlation coefficients between counters and measured energy values. While this approach leads to models with acceptable accuracy, we found that empirically selecting a different (but mostly not disjoint) set of counters yields better results. Furthermore, it is also dependent on the final objective of the models (i.e. minimized mean error vs. minimized max error). Table II lists

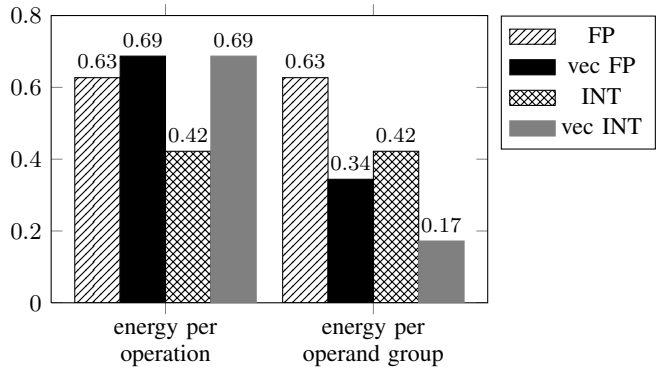

Fig. 1. Energy consumption in nanojoules (nJ) per FMA (including necessary L1 loads and stores) and per group of operands for vectorized and non-vectorized double-precision floating point and integer loads.

all hardware events that we initially considered as input for our model. The selection was done with regard to the capabilities of the hardware and compilers in use.

Although our models are trained with high-level benchmarks from the NAS suite, we implemented 4 fused multiply-add (FMA) micro-benchmarks to get a better grasp on the energy and power consumption characteristics of integer and double-precision (DP) floating point operations, both vectorized and non-vectorized. We chose FMA as the operation to be performed, since it is an expensive operation in terms of the numbers of transistors involved and likely among the execution unit instructions with the highest energy consumption. Each benchmark computes FMAs on one kilobyte of data of the respective type, hence all data resides in the L1 cache. To obtain stable results the computation was performed in a loop to ensure a total operand group count of ~100 billion resulting in a run time of at least 2 seconds. A group of operands denotes all operands required for one non-vector FMA operation, i.e. A, B and C denote one operand group in $C = C + A * B$. Figure 1 illustrates the average E_{dynamic} consumed for a single operation of the respective type, including all necessary operations such as loads/stores from L1. The figure shows that non-vectorized floating point operations can consume up to 49% more energy than non-vectorized integer operations. Furthermore, the vectorized version of the floating point benchmark consumes 45% less per-operand-group energy (10% more per operation) than the non-vectorized version. The same holds for the integer benchmarks, with a reduction of 60% per-operand-group energy (63% more per operation). The unequal per-operand-group differences between the (non-)vectorized floating point and integer cases are explained by the respective vector width for the DP floating point and integer cases. As described in Section III the POWER7 can perform 4 integer but only 2 DP floating point FMAs simultaneously.

Since the impact of integer and floating point instructions on energy consumption differs greatly, we will always include them separately in all our modeling efforts rather than combining them into a single value or just counting all finished instructions. Moreover, because vector operations differ from non-vector operations in the same manner, we consider including them separately whenever they are in use. However, in our setup – contrary to XL – the GNU compiler did not vectorize most portions of the codes. For this reason, counting vector instructions can be omitted when modeling our GCC-compiled benchmarks and applications. For XLC however, we will include them in our efforts, as Figure 1 shows large differences in energy consumption per operation (we consider vector loads as well as float-point and integer vector instructions). Overall, this small experiment illustrates the obvious importance of the instruction mix when selecting appropriate events to be counted for energy modeling.

Counting only finished instructions of execution units can yield high-error results since we do not yet account for any CPU backend stalls due to events such as branch mispredictions or cache misses. Also, rejected instructions are not accounted for since events such as PAPI_FP_INS only count finished floating point instructions, but not necessarily all that were issued. For this reason, we include PM_CMPLU_STALL in our model, which counts cycles for which a hardware thread was stalled due to any reason. At this point it should be noted that while the floating point and integer events count instructions, events such as PM_CMPLU_STALL count cycles instead. Furthermore, the events listed in Table II partially belong to different domains (thread, core and chip) and care must be taken to scale them accordingly for detailed analyses. However, since linear regression inherently computes individual coefficients for all independent variables, we do not have to employ any additional normalization or scaling.

We call the set of input variables common to the GCC and XLC cases (conventional non-vectorized integer and floating point counts via PAPI_INT_INS and PAPI_FP_INS, cycles with stalls via PM_CMPLU_STALL) our *default* configuration. We will use this as the basis for all model configurations, adding additional counters such as vector instructions only as it becomes necessary.

In addition to the POWER7's integer units, also the load/store units (LSUs) are capable of executing simple integer instructions and both GCC and XLC make use of this functionality. A preliminary analysis showed that, for our training benchmarks, GCC uses this feature for up to 7% of the overall integer instructions executed, while XLC reaches 8%. We choose to include this in our evaluation of counters to be selected for linear regression (countable via the hardware event PM_LSU_FX_FIN). For distinction purposes, we call conventional integer instructions *FXU integer instructions* and those executed by the load/store units *LSU integer instructions*. However, we do not expect large improvements by including LSU integer instructions in our model. The low latency of LSU instructions as well as low numbers of LSU integer instructions compared to the total number of instructions will likely limit the impact of this instruction on the overall energy consumption.

VI. Results

As already discussed in Section V we always train our models with 14 of the 15 NAS benchmarks and validate with the 15th and two real-world applications. For this reason, a complete definition of a model instance consists of:

- The idle energy, which is based on the measured idle power consumption of 43.2 watts. Since we found this to be constant also over SMT mode changes, it is not listed individually.

TABLE III. DETAILED RELATIVE MODEL ERRORS IN PERCENT FOR GCC4 AND THE DEFAULT HARDWARE COUNTER SET.

	error					
validation code	mean (NAS)	median (NAS)	max (NAS)	valid. code	MP2C	SHS
dc.W	1.09	0.78	4.40	0.35	0.10	2.70
cg.A	1.08	0.80	4.39	1.21	0.07	2.29
bt.W	1.08	0.80	4.39	0.80	0.06	1.99
lu.W	1.09	0.78	4.41	4.41	0.06	1.76
sp.W	1.08	0.80	4.39	0.16	0.05	1.57
ua.W	1.08	0.80	4.39	0.27	0.04	1.42
ep.W	1.09	0.79	4.39	0.93	0.04	1.30
ft.A	1.08	0.80	4.39	0.76	0.03	1.19
is.B	1.09	0.79	4.39	0.44	0.04	1.10
ua.A	1.03	0.80	4.31	1.03	0.24	1.17
bt.A	1.18	0.74	3.73	3.73	0.32	0.88
sp.A	1.08	0.75	4.64	1.70	0.06	0.98
lu.A	1.16	0.76	4.97	2.85	0.05	0.89
ep.A	1.36	0.96	4.54	2.65	0.40	0.61
ft.B	1.13	0.85	4.25	0.84	0.04	0.79

- The 14 NAS benchmark-problem size combinations used for training. Since it is sufficient to specify the 15th code used for validation to define the other 14, we will use this naming scheme for brevity.
- The hardware counter events used as independent variables for the linear regression part of our model.

To motivate the investigation of energy modeling for different compilers, Figure 2 presents the relative time and energy overhead of GCC8 compared to XLC8. It clearly shows that the compilers differ not only in their ability to produce fast code but also in the caused energy consumption. Moreover, the differences in time and energy are not always correlated. For this reason, we will evaluate our models with GCC and XLC. In addition, we investigate varying parallelism as listed in Table I. All errors given denote relative errors in percent.

A. GCC

First we investigate the results for GCC. Table III shows the results for GCC4 when using the default set of counters, i.e. floating point and FXU integer instructions as well as CPU stalls. Each row represents a model instance with different sets of training benchmarks. The table lists statistical measures for the errors of the training benchmarks as well as the exact errors of the 15th validation code and both applications. As can be seen, using the default configuration already results in high accuracy. The model with the worst maximum error is the one lacking lu.W in the training set, suggesting that it might be a cornerstone in our training suite that cannot be substituted by any other benchmarks, including its larger problem size pendant lu.A. Also, the configuration omitting lu.A shows the highest maximum error of 4.97%, even though lu.A itself can be modeled with only 2.85% error. Overall, we achieve a mean error of 1.11% with the default counter selection.

As discussed in Section V, vector instructions do not occur with our GCC setup, hence they are not included in any configuration pertaining to GCC. However, we also tested adding LSU integer instruction counts to the default set of counters, resulting in almost no error change (on average 0.1%) and being well below our energy measurement accuracy. For this reason and due to space constraints, more detailed results for these configurations are not shown and we recommend adhering to the (smaller) default set of counters.

When switching the hardware to SMT1 mode, the error increases slightly (below 0.1% change in training and for validation of the NAS codes, increase of 1% mean and 1.7% max for MP2C and 0.2% mean and 0.5% max for SHS) due to a higher variation in the measurement results. Since there are no idle hardware threads in SMT1 mode, the operating system causes context switches which are not necessary in SMT4. For brevity, we omit more detailed results for this case.

Results for GCC8 and GCC16 show the same behavior when adding LSU integer instructions (average error changes of 0.3% for GCC8 and 0.5% for GCC16) and are therefore only presented in summary in Table IV. The only noteworthy phenomenon is the dc.W configuration which poses an outlier in the GCC8 case. Here, the validation of dc.W fails with the max error actually increasing by 8.8% percentage points when we include LSU integer instructions. This could be contributed to the fact that dc.W is unusually integer-heavy with integer instructions making up ~75% of the overall instructions executed, and only executes ~50 floating point instructions. Since there is no other benchmark exhibiting such a high relative integer load, it cannot be modeled accurately. Still, the fact that GCC4 and GCC16 do not show this effect leads us to assume that multi-threading on the POWER7 affects the usage of the LSU integer functionality or its contribution to the energy consumption.

A likely reason for the overall lack of improvement when including LSU integer instructions is that their number shows a very high linear correlation (on average 0.997 over all GCC cases) with the number of FXU integer instructions. This, coupled with the fact that GCC only uses the LSU for approximately 7% of the overall integer instructions, confirms our expected low contribution to the overall energy consumption.

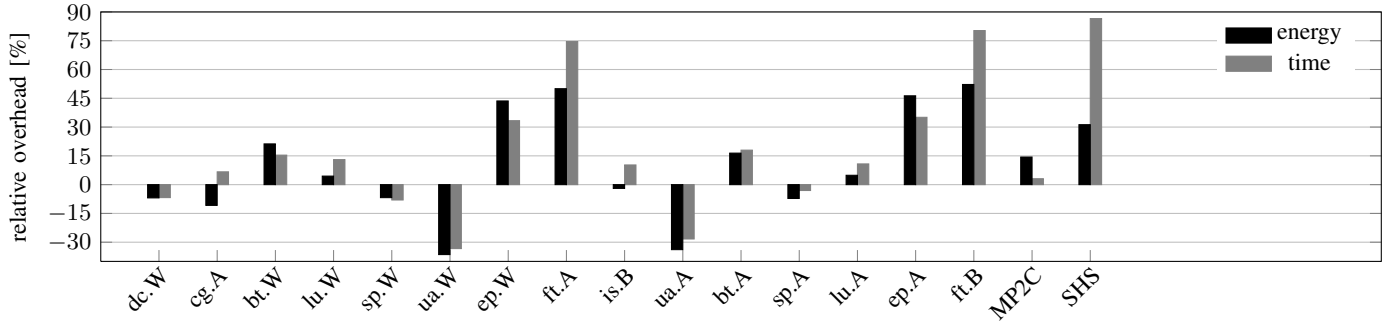

Fig. 2. Relative energy and time overhead of GCC8 over XLC8.

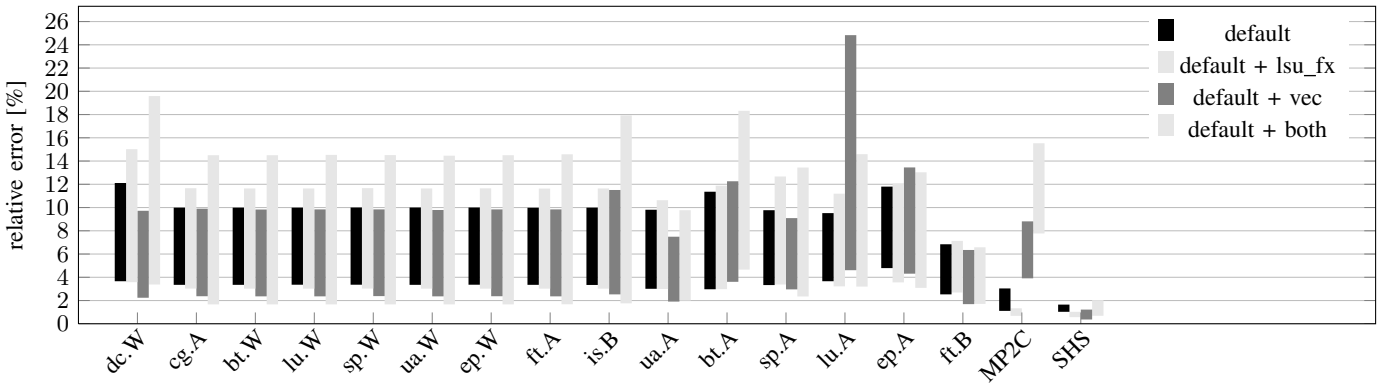

Fig. 3. Error bars depicting mean (lower edge) and max (upper edge) relative errors in percent over all training codes for each model when using only the default input, adding LSU integer counts, vector counts, or both, for the XLC4 case. The two clusters of bars for MP2C and SHS show their respective errors over all model configurations.

B. XLC

We also tested our modeling efforts with the IBM XL compiler, and found it to be more difficult to handle. Contrary to GCC, the IBM compiler makes use of vector instructions and also shows a 15% higher amount of LSU integer instructions compared to GCC. For this reason we will present results for the default selection as well as when taking this extended hardware usage into account.

First, we present the results of the XLC4 case in Figure 3. It shows that with the default counter selection we reach a mean error of 3.1%, with both applications being modeled very well (around 1% error). When adding the LSU integer count, we are able to decrease the error by approximately 0.6 percentage points. We expected this small improvement since the overall usage of this kind of instruction is comparable to GCC's and the GCC results showed a similar effect.

Adding vector instructions instead gives us an average improvement of ~0.8 percentage points, however as illustrated by the figure, our models do not always benefit from this modification. In the case when validating with bt.A or lu.A the average error actually increases. Furthermore, when we use the full set of counters including also vector counts, the average error again decreases by almost 1 percentage point, however the mean error of MP2C increases drastically to 7.8%. This effect shares a similarity with the dc.W phenomenon in the GCC8 case. MP2C is not only the code with the highest amount of vectorization (approximately 14% of all floating point operations are vectorized), it is also the only code – together with SHS – that uses vectorized fused-multiply-add (FMA) instructions. As previously mentioned, vectorized FMAs are an expensive operation in the number of transistors involved. Hence, they are likely among the execution unit instructions with the highest energy consumption and none of our training benchmarks use them. As a result, although the POWER7 can count vectorized FMAs, we are unable to model them. This is one of the rare cases when adding specific micro-benchmarks to our training set could improve the results.

Despite the mean error decreasing overall, the max error increases when adding these counters since we always have 1-2 outliers. First, dc.W again cannot be substituted by any other code, hence it always shows a validation error between 9 and 19% when not included in training. Second, is.B is one of the benchmarks that uses the LSU integer functionality the most. As such, it is also irreplaceable and as an outlier responsible for the fairly constant max error of 13-15%.

Comparable to our GCC scenario, switching off SMT (i.e. using SMT1 mode instead of SMT4) also shows little effect for XLC. Hence, for the XLC4_NOSMT case compared to XLC4, we observe an error decrease of 0.1% mean and 1%

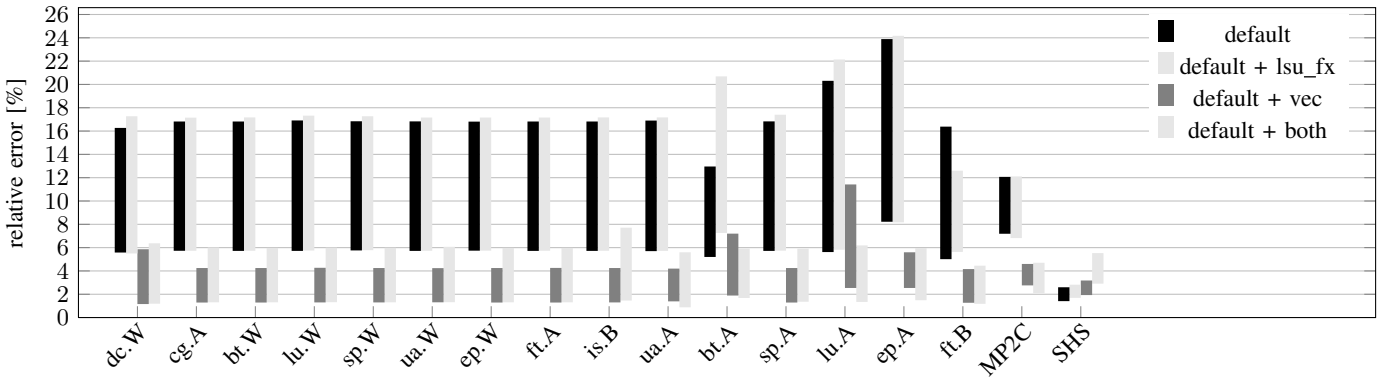

Fig. 4. Error bars depicting mean (lower edge) and max (upper edge) relative errors in percent over all training codes for each model when using only the default input, adding LSU integer counts, vector counts, or both, for the XLC8 case. The two clusters of bars for MP2C and SHS show their respective errors over all model configurations.

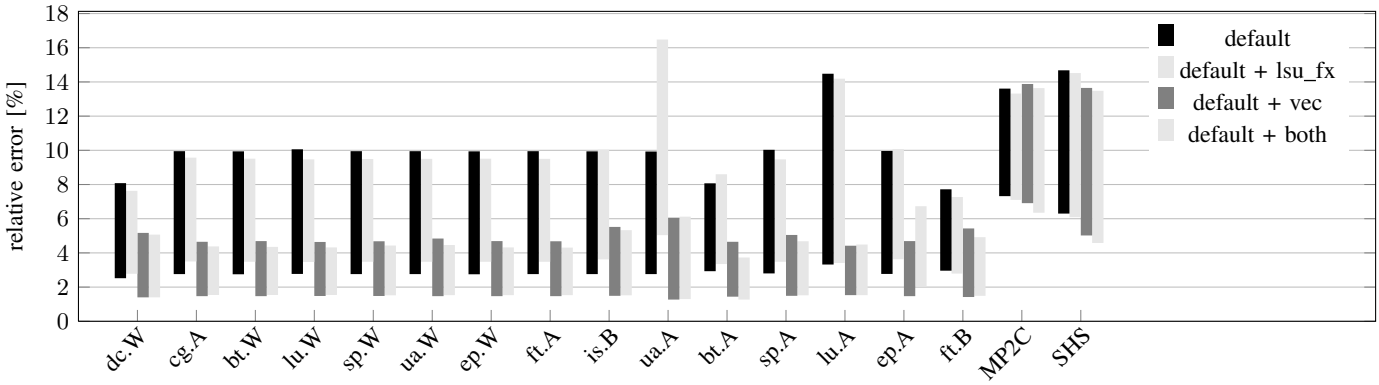

Fig. 5. Error bars depicting mean (lower edge) and max (upper edge) relative errors in percent over all training codes for each model when using only the default input, adding LSU integer counts, vector counts, or both, for the XLC16 case. The two clusters of bars for MP2C and SHS show their respective errors over all model configurations.

max error for the NAS codes, as well as an increase for MP2C (0.5% mean and 0.8% max) and virtually no change for SHS (less than 0.1% mean and max).

As illustrated by Figure 4 however, results are improving when moving to the XLC8 case (using two hardware threads per core). While the effect of the LSU integer instruction count is almost negligible, including vector counts greatly improves the accuracy of our models from an average mean error of 5.6% to 1.6%. The fact that XLC8 exhibits effects that are inverse to our observations for XLC4 once again shows the significance of including multi-threading in energy modeling efforts. The lowest overall mean error is achieved when including the full set, default + LSU integer + vector instructions. The results for the max error show a similar behavior, with a decrease from 16% for the default set to below 5% for default + vector counts. Adding LSU integer counts again has a very small effect, however it increases the average max error again to 16.7% and 5.9% respectively.

Results for the XLC16 case, depicted in Figure 5, show very similar behavior. The best average mean error (1.98%) is achieved when adding both LSU counts and vector counts, and the best average max error (5.8%) is also achieved by using the full set. However, the figure clearly shows that the results for our two application codes are significantly worse compared to the NAS codes. While adding counts for the LSU and vector units reduce the mean error by approximately 1-2 percentage points, the maximum error remains fairly constant at roughly 14%. We do not have an explanation for this phenomenon as of yet and experiments involving other counters from our initial selection listed in Table II did not result in any significant improvement. However, the fact that this phenomenon does not occur for XLC8 and XLC4 again leads us to assume that either multi-threading affects the usage of vector or LSU instructions (or their energy consumption), or that such effects caused by SMT are not accounted for in our initial selection of hardware counters.

TABLE IV. OVERALL RELATIVE MODEL ERRORS IN PERCENT FOR ALL CASES.

setup	codes	mean	median	max
GCC4_NOSMT	NAS	1.12	0.98	4.99
	MP2C	1.27	1.22	2.14
	SHS	1.66	1.44	3.25
GCC4	NAS	1.11	0.80	4.97
	MP2C	0.31	0.11	1.52
	SHS	3.31	3.32	3.78
GCC8	NAS	1.62	1.62	5.30
	MP2C	0.22	0.10	1.01
	SHS	2.00	1.70	4.25
GCC16	NAS	1.95	1.92	5.10
	MP2C	0.57	0.54	1.12
	SHS	0.07	0.04	0.27
XLC4_NOSMT	NAS	2.99	1.17	14.03
	MP2C	1.24	1.20	2.11
	SHS	0.95	0.91	1.79
XLC4	NAS	3.10	1.23	15.02
	MP2C	0.68	0.68	1.33
	SHS	0.57	0.53	1.03
XLC8	NAS	1.32	0.58	7.71
	MP2C	2.09	2.04	4.70
	SHS	2.91	2.78	5.53
XLC16	NAS	1.52	0.96	6.73
	MP2C	6.35	5.88	13.64
	SHS	5.32	4.36	13.48

Table IV summarizes the overall results for all our test cases. As a final selection, we use our default counter selection (floating point instruction, FXU integer instruction and stall cycle counts) for all GCC cases, we extend to default + LSU integer instructions for XLC4 and we furthermore add vector instructions for XLC8 and XLC16. To conclude the results analysis, the table shows that while the GCC cases are covered easily, XLC requires not only more counters but also shows higher errors. We assume this to be an effect of XLC's more complex usage of the hardware and knowledge about its characteristics.

VII. CONCLUSION

Within this paper we presented energy models for the POWER7 processor based on linear regression. We have shown that it can be successfully applied to achieve high accuracy when directly modeling CPU energy consumption for various workloads. The key findings of this research comprise:

- the hardware counter selection used as input parameters for linear regression (being compiler- and multi-threading-dependent),
- despite the high complexity of the POWER7 processor, linear regression can achieve high accuracy even with a limited number of input variables depending on the compiler and multi-threading settings, and

- using micro-benchmarks to train energy models can be unnecessary; relying on a sufficiently large number of high-level benchmarks instead can yield energy consumption predictions with high accuracy.

In addition to the research presented, we also investigated memory energy consumption modeling. We chose a reasonable selection of performance counters (memory reads and writes, cache misses, prefetching instructions, etc.) and measured memory energy consumption. However, keeping our methodology, we were unable to achieve any level of accuracy that is suitable for our needs (results showed average errors between 10 and 15%). It is yet unclear whether this is a result of our simplistic approach or the lack of suitable hardware counters to capture all characteristics necessary for this task.

Possible future work extending this research includes the aforementioned memory modeling as well as improving the results for the XLC cases and multi-threading or developing a more complex, unified model for all use cases of compilers. Alternatively, the distinction between compilers could be removed by statically investigating the instruction mix and using this data as a basis for selecting regression input. Moreover, the model's granularity could be increased to a per-core or per-thread basis. Furthermore, support for DVFS could be added or the effect of the POWER7's various prefetching settings on energy modeling investigated.

Finally, we plan to integrate our models into existing software such as the Scalasca performance analysis tool [5] (to provide energy data without the need for additional hardware to run Amester) as well as the Insieme Compiler [4] (to perform multi-objective auto-tuning that includes energy consumption minimization as an objective).

ACKNOWLEDGEMENTS

The authors would like to thank Charles Lefurgy for his assistance in working with Amester and insightful discussions throughout the course of this research. The results presented were obtained using the IBM Automated Measurement of Systems for Temperature and Energy Reporting software. This work was partially supported by the Austrian Science Fund (FWF) DK-plus project Computational Interdisciplinary Modelling, W1227-N16.

REFERENCES

[1] M. Tolentino and K. W. Cameron, "The optimist, the pessimist, and the global race to exascale in 20 megawatts," *Computer*, vol. 45, no. 1, pp. 95–97, 2012.

[2] D. Bedard, R. Fowler, M. Linn, and A. Porterfield, "PowerMon 2: Fine-grained, integrated power measurement," *Renaissance Computing Institute, Tech. Rep. TR-09-04*, 2009.

[3] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron, "Powerpack: Energy profiling and analysis of high-performance systems and applications," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 5, pp. 658–671, 2010.

[4] H. Jordan, P. Thoman, J. J. Durillo, S. Pellegrini, P. Gschwandtner, T. Fahringer, and H. Moritsch, "A multi-objective auto-tuning framework for parallel codes," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*. IEEE, 2012, pp. 1–12.

[5] M. Geimer, F. Wolf, B. J. Wylie, E. Ábrahám, D. Becker, and B. Mohr, "The Scalasca performance toolset architecture," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 702–719, 2010.

[6] G. Kestor, R. Gioiosa, D. Kerbyson, and A. Hoisie, "Enabling accurate power profiling of HPC applications on exascale systems," in *Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers*. ACM, 2013, p. 4.

[7] B. Goel, S. A. McKee, R. Gioiosa, K. Singh, M. Bhadauria, and M. Cesati, "Portable, scalable, per-core power estimation for intelligent resource management," in *Green Computing Conference, 2010 International*. IEEE, 2010, pp. 135–146.

[8] M. Y. Lim, A. Porterfield, and R. Fowler, "SoftPower: fine-grain power estimations using performance counters," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 308–311.

[9] W. Huang, C. Lefurgy, W. Kuk, A. Buyuktosunoglu, M. Floyd, K. Rajamani, M. Allen-Ware, and B. Brock, "Accurate fine-grained processor power proxies," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2012, pp. 224–234.

[10] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: memory power estimation and capping," in *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on*. IEEE, 2010, pp. 189–194.

[11] *BIOS and Kernel Developers Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors*, Advanced Micro Devices, Jan 2013. [Online]. Available: http://support.amd.com/TechDocs/42301_15h_Mod_00h-0Fh_BKDG.pdf

[12] L. Brochard, R. Panda, and S. Vemuganti, "Optimizing performance and energy of HPC applications on POWER7," *Computer Science-Research and Development*, vol. 25, no. 3-4, pp. 135–140, 2010.

[13] M. Knobloch, M. Foszczynski, W. Homberg, D. Pleiter, and H. Böttiger, "Mapping fine-grained power measurements to HPC application runtime characteristics on IBM POWER7," *Computer Science - Research and Development*, pp. 1–9, 2013.

[14] M. Ware, K. Rajamani, M. Floyd, B. Brock, J. C. Rubio, F. Rawson, and J. B. Carter, "Architecting for power management: the IBM® POWER7 approach," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*. IEEE, 2010, pp. 1–11.

[15] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," *ACM SIGARCH Computer Architecture News*, vol. 28, no. 2, pp. 83–94, 2000.

[16] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, 2002.

[17] A. Flores, J. L. Aragon, and M. E. Acacio, "Sim-PowerCMP: a detailed simulator for energy consumption analysis in future embedded CMP architectures," in *Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on*, vol. 1. IEEE, 2007, pp. 752–757.

[18] B. Sinharoy *et al.*, "IBM POWER7 multicore server processor," *IBM Journal of Research and Development*, vol. 55, no. 3, pp. 1–1, 2011.

[19] C. R. Lefurgy, "Questions regarding power measurements on POWER7," e-mail communication, May 2013.

[20] SNU NPB Suite. Accessed: 16.07.2013. [Online]. Available: http://aces.snu.ac.kr/Center_for_Manycore_Programming/SNU_NPB_Suite.html

[21] G. Sutmann, L. Westphal, and M. Bolten, "Particle based simulations of complex systems with MP2C: Hydrodynamics and electrostatics," *AIP Conference Proceedings*, vol. 1281, no. 1, pp. 1768 – 1772, 2010.

[22] L. Arnold, "IBM BG/P workshop," Oct 2009. [Online]. Available: http://www.training.prace-ri.eu/uploads/tx_pracetmo/BlueGeneP.pdf

[23] J. Dongarra, K. London, S. Moore, P. Mucci, and D. Terpstra, "Using PAPI for hardware performance monitoring on Linux systems," in *In Conference on Linux Clusters: The HPC Revolution, Linux Clusters Institute*, 2001.

[24] V. A. Mericas, B. Elkin, and V. R. Indukuru, "Comprehensive PMU event reference - POWER7," 2011.