Delft University of Technology
Parallel and Distributed Systems Report Series

PAM-SoC Experiments and Results

A.L. Varbanescu
A.L.Varbanescu@ewi.tudelft.nl

report number PDS-2006-001

ISSN 1387-2109

Published and produced by:
Parallel and Distributed Systems Section
Faculty of Information Technology and Systems Department of Technical Mathematics and Informatics
Delft University of Technology
Zuidplantsoen 4
2628 BZ Delft
The Netherlands

Information about Parallel and Distributed Systems Report Series:
reports@pds.twi.tudelft.nl

Information about Parallel and Distributed Systems Section:
http://pds.twi.tudelft.nl/

Abstract

This report briefly presents PAM-SoC, a toolchain for predicting MPSoC platform performance, and its results on predicting the performance of a set of six benchmark applications.

1 Introduction

PAM-SoC is a light semi-static performance prediction toolchain that computes system-level performance estimations for applications running on MPSoCs. PAM-SoC is based on PAMELA[6], a static performance prediction methodology for general purpose parallel platforms (GPPPs). By coupling an application model with the target machine model, PAMELA computes the lower bound of the execution time of the application on the target architecture. To address the specifics of MPSoCs, PAM-SoC includes new techniques for machine modeling and tools for gathering memory behavior statistics. In its prediction, PAM-SoC trades accuracy for estimation speed: in cases when cycle-accurate simulation would take tens of minutes, application behavior can be estimated in (tens of) seconds. Thus, PAM-SoC can be part of the design flow of an MPSoC platform, for either application or architecture tuning.

2 PAMELA

PAMELA (PerformAnce ModEling LAnguage) [5] is a performance simulation formalism that facilitates symbolic cost modeling, featuring a modeling language, a compiler, and a performance analysis technique that enables PAMELA models to be compiled into symbolic performance models that trade prediction accuracy for the lowest possible solution complexity. In this symbolic cost modeling, parallel programs implemented using the Series-Parallel (SP) [7] programming paradigm are mapped into explicit, algebraic performance expressions in terms of program parameters (e.g., problem size), and machine parameters (e.g., number of processors). Instead of being simulated, the model is automatically *compiled* into a *symbolic cost model*, that can be further compiled into a *time-domain cost model* and, finally, evaluated into a *time estimate*.

Modeling.

The PAMELA modeling language is a process-oriented language designed to capture concurrency and timing behavior of parallel systems. Data computations from the original source code are modeled into the *application model* in terms of their resource requirements and workload. The available resources and their usage policies are specified by the *machine model*.

Any PAMELA model is written as a set of process equations, composed from `use` and `delay` basic processes, using sequential, parallel, and conditional composition operators. The construct `use(Resource,`t`)` stands for exclusive acquisition of `Resource` for t units of (virtual) time. The construct `delay(`t`)` stands for program execution for t units of (virtual) time (without using potentially shared resources).

Note that the *machine model* is expressed in terms of available resources and an abstract instruction set (AIS) for using these resources. Separate abstract instructions are created for operations that (1) use different resources, (2) use the same resources, but with different latencies (i.e., different use time), or (3) use different parameters. The *application model* of the parallel program is implemented using an (automated) translator from the source instruction set to the machine AIS. The example below illustrates the modeling of a block-wise parallel addition computation $y = \sum_{i=1}^{N} x_i$ on a machine with `P` processors and shared memory `mem`:

```
// application model:                  // machine model
   par (p=1,P) {                          load=use(mem,t_access)
       seq (i=1,N/P) { load ; add } ;     add=delay(t_add)
       store }
```

Symbolic Compilation and Evaluation.

A PAMELA model is translated into a *time-domain performance model* by substituting every process equation by a numeric equation that models the execution time associated with the original process. The result is a new PAMELA model that only comprises numeric equations, as the original `process` and `resource` equations are no longer present. The PAMELA compiler can further reduce and evaluate this model for different numerical values of the parameters, computing the lower bound of the application execution time. The analytic approach underlying the translation, together with the algebraic reduction engine that drastically optimizes the evaluation time, are detailed in [4].

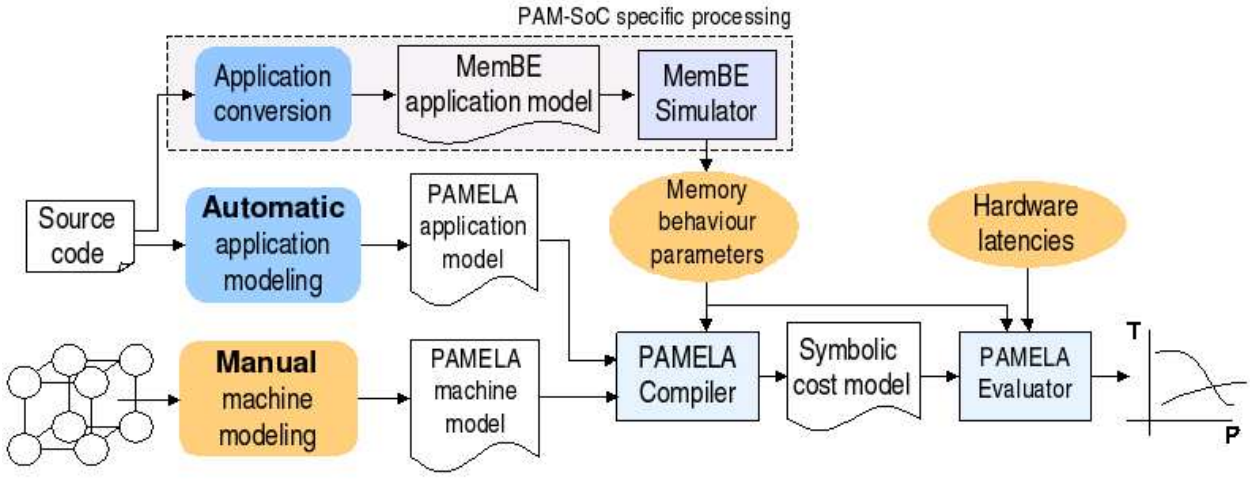


Figure 1: The PAM-SoC toolchain

3 The PAM-SoC toolchain

Using PAMELA for MPSoC performance predictions is quite difficult because of the architecture and application modeling efforts required. Details that can be safely ignored for GPPP models, as they do not have a major influence on the overall performance, may have a significant influence on MPSoC behavior. As a consequence, for correct modeling of MPSoC applications and architectures, we have extended PAMELA with new techniques and additional memory behavior tools. The resulting PAM-SoC toolchain is presented in Figure 1. In this section we will further detail its specific components.

3.1 MPSoC Machine Modeling

The successfull modeling of a machine architecture starts with accurate detection of its important *contention points*, i.e., system resources that may limit performance when used concurrently. Such resources can be modeled at various degrees of detail, i.e., *granularities*, by modeling more or less from their internal sub-resources. The model granularity is an essential parameter in the speed-to-accuracy balance of the prediction: a finer model leads to a more accurate prediction (due to better specification of its contention points), but it is evaluated slower (due to its increased complexity). Thus, a model *granularity boundary* should be established for any architecture, so that the prediction is is still fast and sufficiently accurate. This boundary is usually set empirically and/or based on a set of validation experiments.

Previous GPPPs experiments with PAMELA typically used coarse models, based on three types of system resources: the processing units, the communication channels and the memories. For MPSoC platforms, we have established a new, extended set of resources to be included in the machine model, as seen in Fig. 2. The new granularity boundaries (the leaf-level in the resource tree in Fig. 2) preserve a good speed-to-accuracy balance, while allowing drastic simplification of the MPSoC machine modeling procedure.

Some additional comments with respect to the machine modeling are the following:

- When on-chip programmable processors have subunits able to work in parallel, they should be modeled separately, especially when analyzing applications that specifically stress them.
- The communication channels require no further detailing for shared-bus architectures. For more complex communication systems, involving networks-on-chips or switch boxes, several channels may be acting in parallel. In this case, they have to be detected and modeled separately.
- The memory system is usually based on individual L1's, an L2 shared cache (maybe banked) and off-chip memory (eventually accessed by dedicated L2 Refill and L2 Victimize engines). If hardware snooping coherency is enforced, two more dedicated modules become of importance: the Snooping and

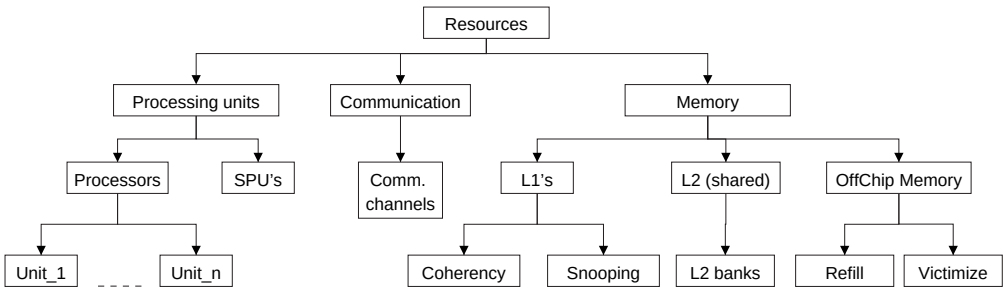


Figure 2: The extended set of resources to be included in a PAM-SoC machine model

the Coherency engines. Any of these components present in the architecture must be also included in the model.

After identifying model resources, the AIS has to be specified as a set of rules for using these resources, requiring (1) in-depth knowledge on the functionality of the architecture, for detecting the resource accesses an instruction performs, and (2) resource latencies. As an example, Table 1 presents a snippet from a (possible) AIS, considering an architecture with: several identical programmable processors (`Procs(p)`), each one having parallel arithmetic (`ALU(p)`) and multiplication (`MUL(p)`) units and its own `L1(p)` cache; several Specialized Functional Units (`SFU(s)`); a shared `L2` cache, banked, with dedicated `Refill` and `Victimize` engines; virtually infinite off-chip memory (`mem`).

Table 1: Snippet from an AIS for a generic MPSoC

Operation	**Model**
p: ADD	`use(ALU(p),` t_{ADD}`)`
p: MUL	`use(MUL(p),` t_{MUL}`)`
s: EXEC	`use(SFU(s),` t_{SFU}`)`
p: accessL1(addr)	`use(L1(p),` $t_{L1}^{hit} * h_{L1}^{ratio} + t_{L1}^{miss} * (1 - h_{L1}^{ratio})$`)`
p: accessL2(addr)	`use(L2(bank(addr)),` $t_{L2}^{hit} * h_{L2}^{ratio} + t_{miss} * (1 - h_{L2}^{ratio})$`)`
p: refillL2(addr)	`use(Refill,` t_{Mem}^{RD} `)`
p: victimizeL2(addr)	`use(Victimize,` $victimization^{ratio} * t_{Mem}^{WR}$ `)`

p: READ(addr)	`accessL1(addr);`
	`if (missL1) { accessL2(addr);`
	`if (missL2) {`
	`if (victimize) victimizeL2(addr);`
	`refillL2(addr)}}`

The cache hit/miss behavior cannot be evaluated using the cache (directory) state, because Pamela, being algebraic, is a state-less formalism. Thus, we compute a probabilistic average cache latency, depending on the cache hit ratio, h^{ratio}, and on the hit/miss latencies, t^{hit} and t^{miss}. Also the `if` branches in the `READ(addr)` model are addressed in a probabilistic manner. For example, `if(missL1)` is replaced by a quantification with $(1 - h_{L1}^{ratio})$, which is the probability that this condition is true. All these probabilistic models are based on *memory behavior parameters* which are both application- and architecture-dependent. PAM-SoC uses an additional tool for computing these parameters, which is presented in Section 3.3.

3.2 Application modeling

Translating an application implemented in a high-level programming language to its Pamela application model (as well as writing a Pamela model from scratch) implies two distinct phases: (1) modeling the application as a series-parallel graph of processes, and (2) modeling each of the processes in terms of Pamela machine instructions. However, modeling an existing application to its Pamela model is a translation

from one instruction set to another, and it can be automated if both instruction sets are fully specified as exemplified in [6].

3.3 The numerical parameters

For computing its prediction, PAM-SoC uses two types of numerical parameters: (1) the hardware latencies (measured under no-contention conditions), and (2) the caches hit ratios. While the former have been also required by GPPP models, the latter become of importance mainly for modeling MPSoC platforms.
The hardware latencies are fixed values for a given architecture and can be either obtained from the hardware specification itself (i.e., theoretical latencies) or by means of micro-benchmarking (i.e., measured latencies). We have based our experiments on the theoretical latencies, aiming to keep the ratio's between latency categories (e.g., between an addition operation and an L1 access) as close as possible to that of real components. The hit ratio's are both machine- and application-dependent, and they have to be computed/evaluated on a per-application basis. For this, we have built MemBE, a custom light-weight *Memory system Behavior Emulator* able to obtain these memory parameters and other various memory statistics with good speed and satisfactory accuracy. MemBE is built as a multi-threaded application that permits the (re)configuring of a custom memory hierarchy using the memory components supported by PAM-SoC. MemBE emulates the memory system of the target architecture and executes a simplified version of the analyzed application[1] (that only includes its memory references), performing no data-processing, but carefully monitoring the data-path of the application.

4 Experiments and results

The validation process of PAM-SoC aims to prove its abilities to correctly predict application behavior on a given MPSoC platform. For these experiments, we have modeled the Wasabi platform, one tile of the CAKE architecture from Philips [8, 1]. A Wasabi chip, presented in Figure **??**, is a shared-memory MPSoC, having 1-8 programmable processors, several SFUs, and various interfaces. The tile memory hierarchy has three levels: (1) private L1's for each processor, (2) one shared on-chip L2 cache, available to all processors, and (3) one off-chip memory module. Hardware consistency between all L1's and L2 is enforced. For software support, Wasabi runs eCos[2], a modular open-source Real-Time Operating System (RTOS), which has embedded support for multithreading. Programming is done in C/C++, using eCos synchronization system calls and the default eCos thread scheduler.

The simulation experiments have been run on Wasabi's configurable cycle-accurate simulator, provided by Philips. For our experiments, we have chosen a fixed memory configuration (L1's are 256KB, L2 is 2MB, and the off-chip memory is 256MB) and we have used up to 8 identical Trimedia processors[2].

For validation, we have implemented a set of six simple benchmark applications, each of them being a possible component of a more complex, real-life MPSoC application. All benchmark applications have been implemented for shared-memory (to comply with the Wasabi memory system), using the SP-programming model and exploiting data-parallelism only (no task parallelism, which is a natural choice for the case of these one-task applications). To test PAM-SoC's abilities, we have compared the measured speed-up (as given by the simulator) and the predicted speed-up (as computed by PAM-SoC) of each of these applications for Wasabi instances with 1 up to 8 processors, and for several input data sizes. The comparison results, in terms of graphs, accuracy and execution times are presented below.

Matrix addition - a memory intensive application - see Fig.3.

Matrix multiplication - a computation intensive application - see Fig. 4.

RGB-to-YIQ conversion - a color-space transformation filter, part of the EEMBC Consumer suite[3] - see Fig. 5.

[1] Currently, the application simplification from the source code to its MemBE model is done by hand. In principle, we believe that it is possible for Pamela and MemBE to use a common application model.

[2] TriMedia is a family of Philips VLIW processors optimized for multimedia processing

Table 2: Simulation vs. prediction times [s]

Application	Data size	T_{sim}	T_{MemBE}	T_{Pam}	T_{PAMSoC}	Speed-up
MADD	3x1024x1024 words	94	2	1	3	31.3
MMUL	3x512x512 words	8366	310	2	312	26.8
RGB-to-YIQ	6x1120x840 bytes	90	7	4	11	8.1
RGB-to-Grey	4x1120x840 bytes	62	3	1	4	15.5
Grey-Filter	2x1120x840 bytes	113	6	4	10	11.3
Filter chain	8x1120x840 bytes	347	20	12	32	10.8

RGB-to-Grey conversion - another color-space transformation - see Fig. 6.

High-pass Grey filter - an image convolution filter, part of the EEMBC Digital Entertainment suite [3] - see Fig. 7.

Filter chain - a chain of three filters (YIQ-to-RGB, RGB-to-Grey, High-pass Grey) successively applied on the same input data - see Fig. 8.

For all the applications, the behavior trend is correctly predicted by PAM-SoC. The average error between simulation and prediction is within 19%, while the maximum is less than 25% (excluding simulator anomalies, like those visible for matrix addition with matrices 256x256 elements). These deviations are due to (1) the differences between the theoretical Wasabi latencies and the ones implemented in the simulator (50-70%), (2) the averaging of the memory behavior data, and (3) the PAMELA accuracy-for-speed trade. Table 2 presents the significant speed-up of PAM-SoC prediction time ($T_{PAMSoC} = T_{MemBE} + T_{Pam}$) compared to the cycle-accurate simulation time, T_{sim}, for the considered benchmark applications and the largest data sets we have measured. While a further increase of the data set size leads to a significant increase for T_{sim} (tens of minutes), it has a minor impact on T_{Pam} (seconds) and leads to a moderate increase of T_{MemBE} (tens of seconds up to minutes). Because MemBE is at its first version, there is still much room for improvement, by porting it on a parallel machine and/or by including more aggressive optimizations. In the future, alternative cache simulators or analytical methods (when/if available) may even replace MemBE for computing memory statistics.

5 Conclusions and Future Work

In this report we have introduced PAM-SoC, a semi-static performance prediction toolchain for MPSoC's. Static performance prediction for MPSoCs is motivated by its reduced cost, which allows it to be a part of the design loop. Even though static performance predictors trade accuracy for estimation cost, a behavior estimation within minutes is more valuable, in the early design phases, than a twenty-hours long simulation with very precise results. We have shown how PAM-SoC is used to predict the performance of MPSoC platforms. To validate the methodology, we have modeled a real MPSoC platform and compared the PAM-SoC prediction results for a set of six simple benchmark applications with the simulation results. These results are fully presented. For the future, we plan to explore three directions: (1) to model and test more complex applications, for further validation/improvement of PAM-SoC, (2) to make the modeling process as automatic as possible, and (3) to investigate whether PAM-SoC can become a truly static performance predictor.

References

[1] D. Borodin. Optimisation of multimedia applications for the Philips Wasabi multiprocessor system. Master's thesis, TU Delft, 2005.

[2] eCos Home page. `http://ecos.sourceware.org/`.

[3] E.E.M.B.C Benchmarks. `http://www.eembc.org/benchmark/`.

[4] H. Gautama and A. v. Gemund. Static performance prediction of data-dependent programs. In *Proc. WOSP'00*. ACM.

[5] A. v. Gemund. *Performance Modeling of Parallel Systems*. PhD thesis, Delft University of Technology, April 1996.

[6] A. v. Gemund. Symbolic performance modeling of parallel systems. *IEEE TPDS*, February 2003.

[7] A. Gonzalez-Escribano. *Synchronization Architecture in Parallel Programming Models*. PhD thesis, Dpto. Informatica, University of Valladolid, 2003.

[8] P. Stravers and J. Hoogerbrugge. Homogeneous multiprocessing and the future of silicon design paradigms. In *Proc. VLSI-TSA'01*, April 2001.

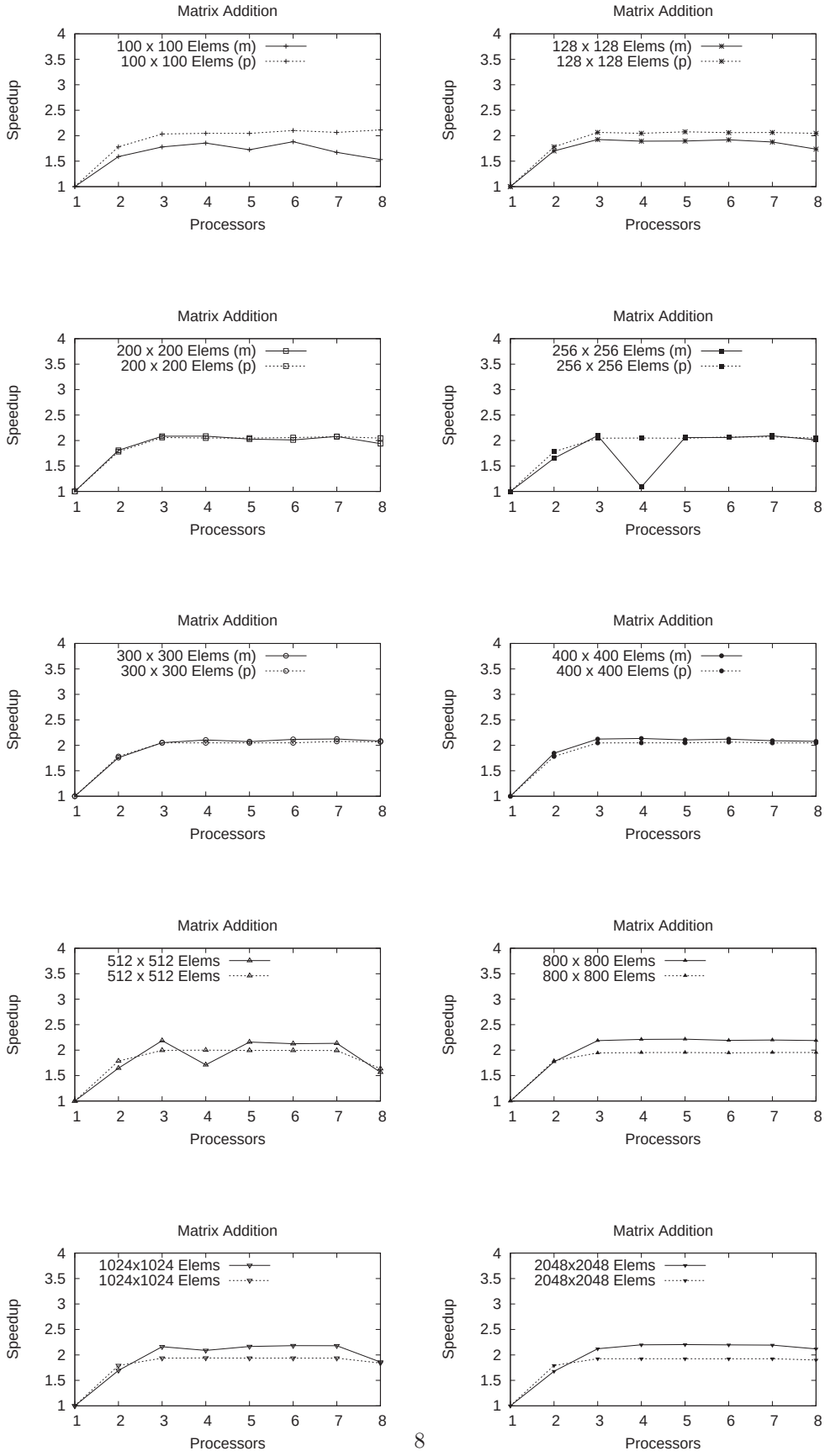


Figure 3: Matrix addition measurement vs. prediction

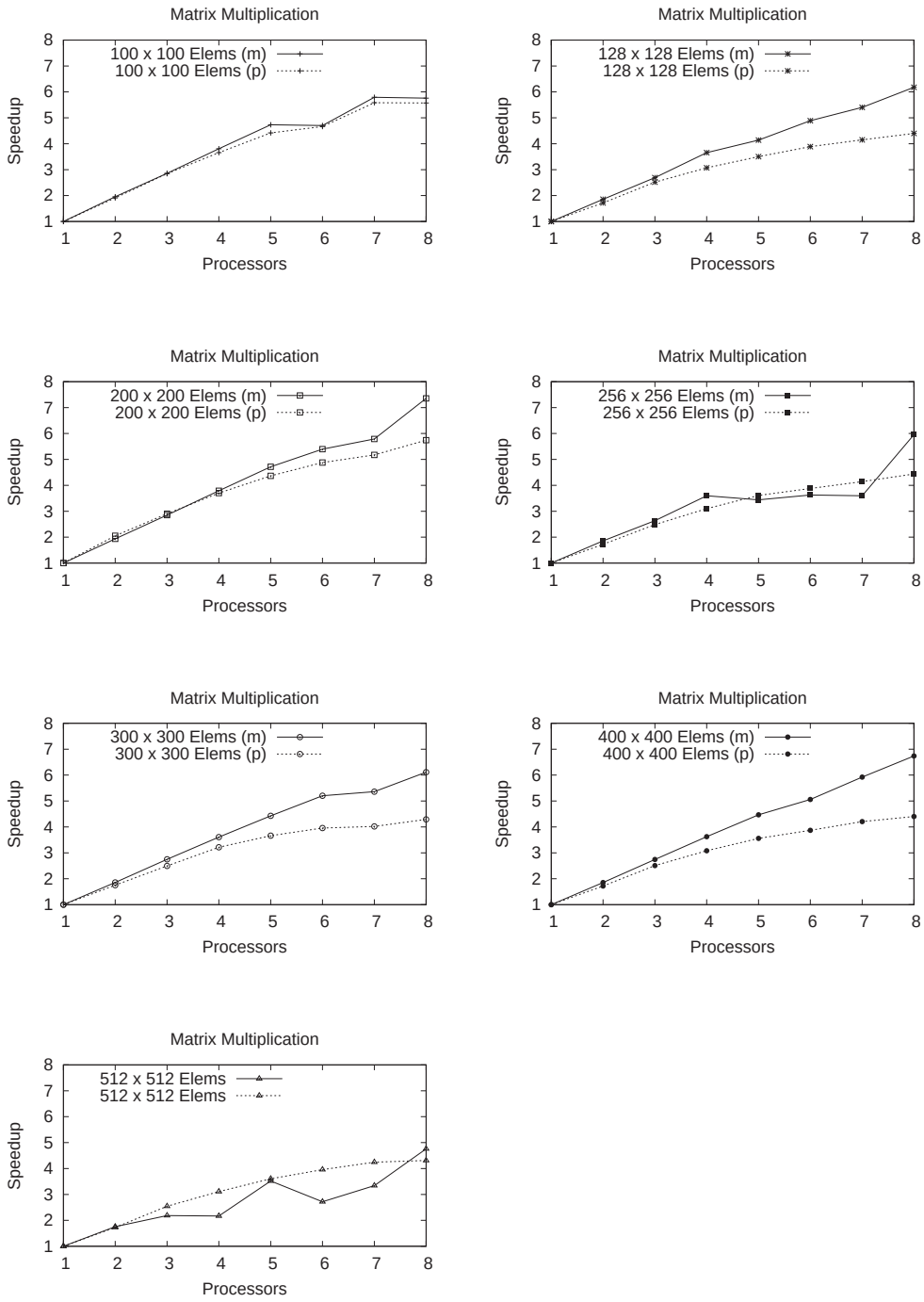


Figure 4: Matrix multiplication measurement vs. prediction

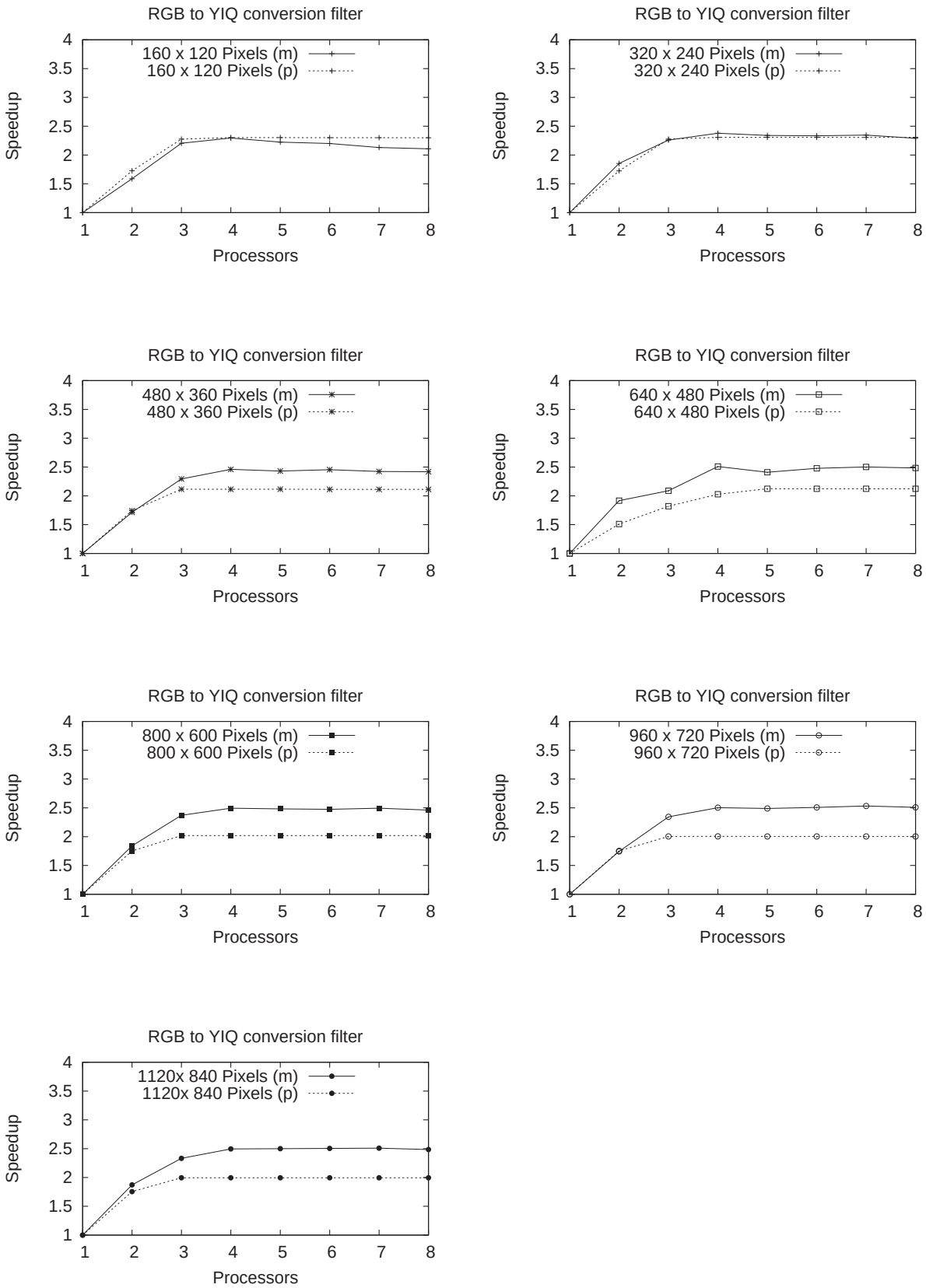


Figure 5: RGB to Grey scale conversion filter measurement vs. prediction

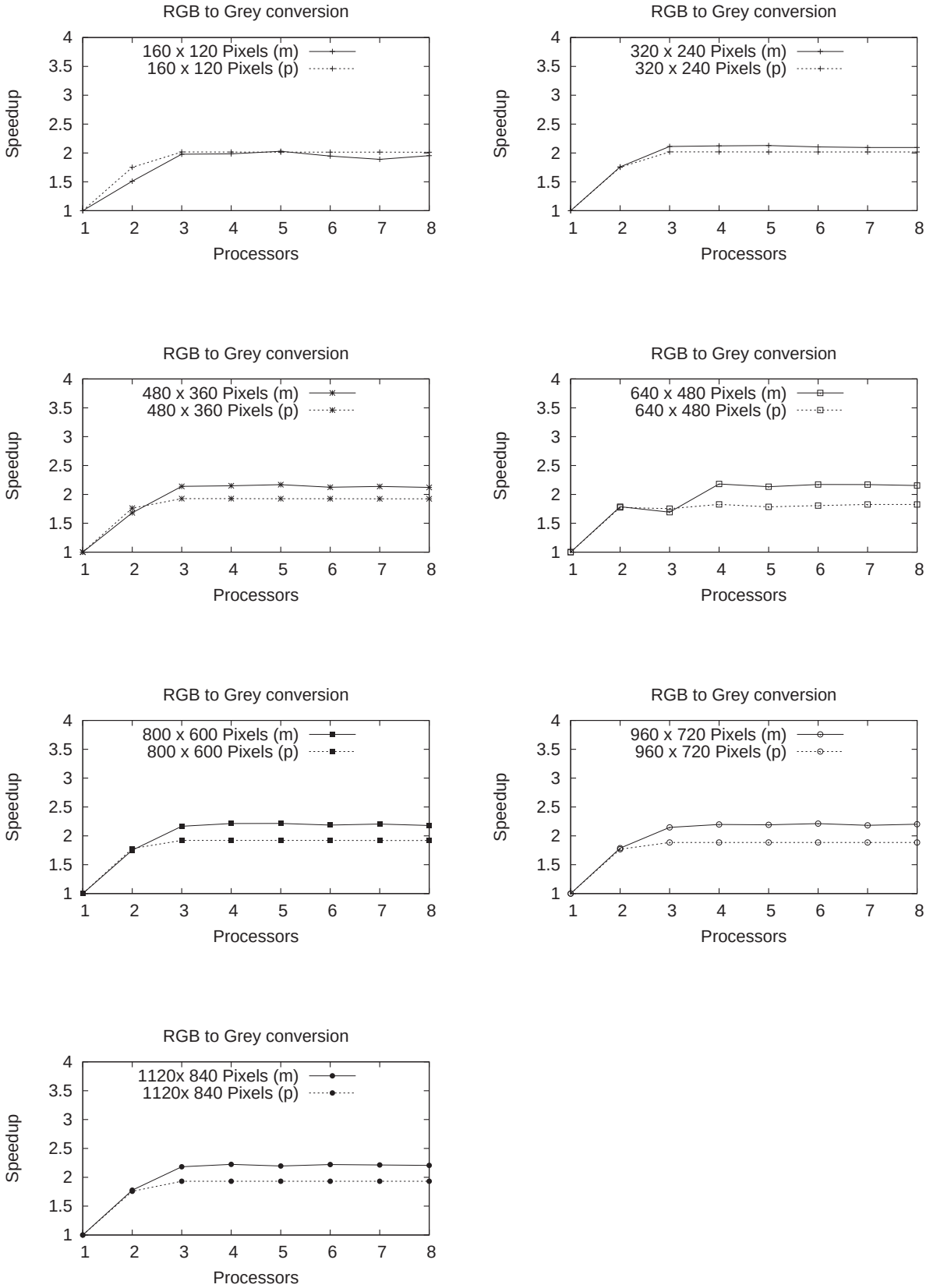
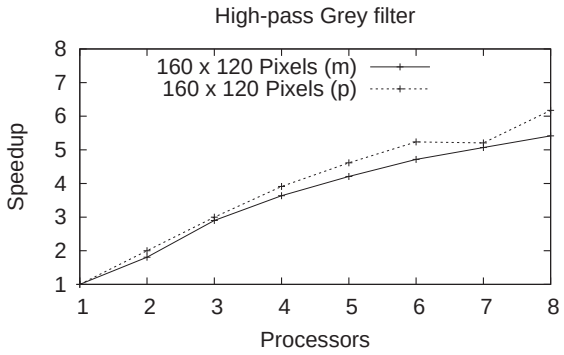
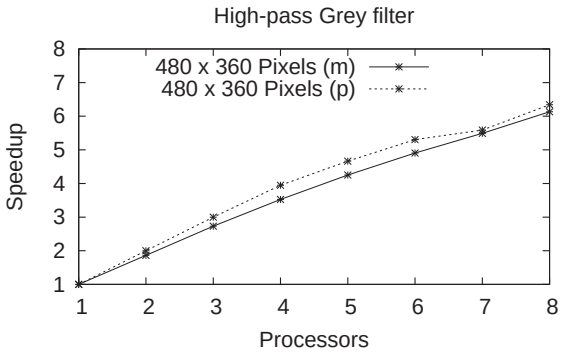
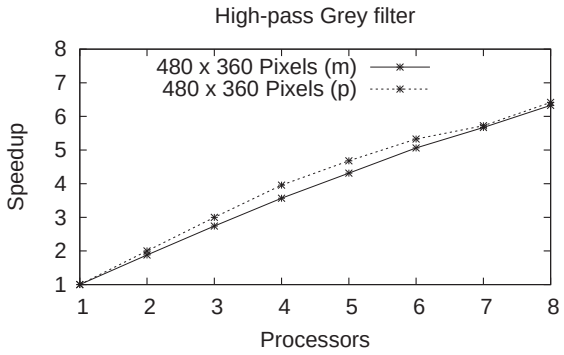
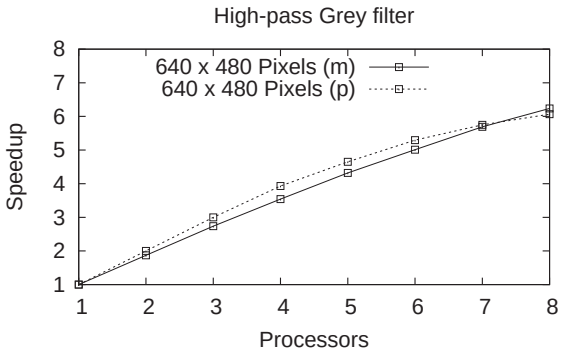


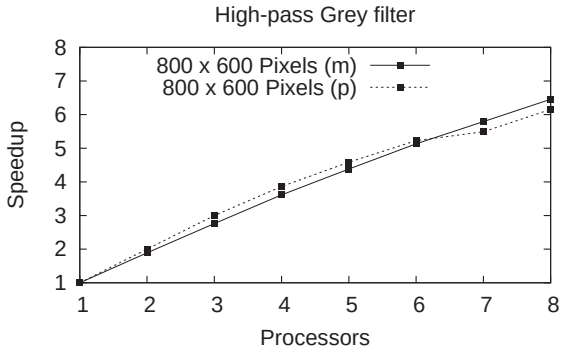
Figure 6: RGB to Grey scale conversion filter measurement vs. prediction

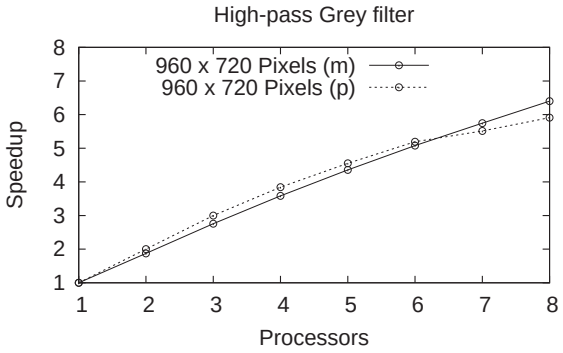












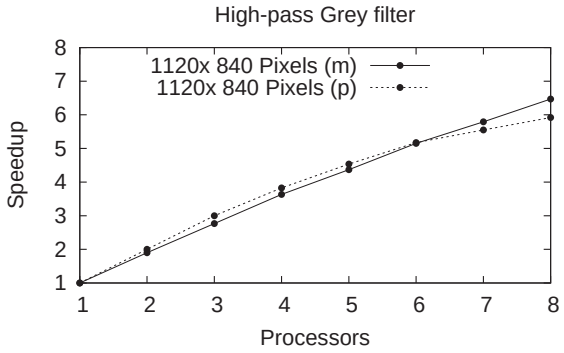


Figure 7: High-pass Grey filter measurement vs. prediction

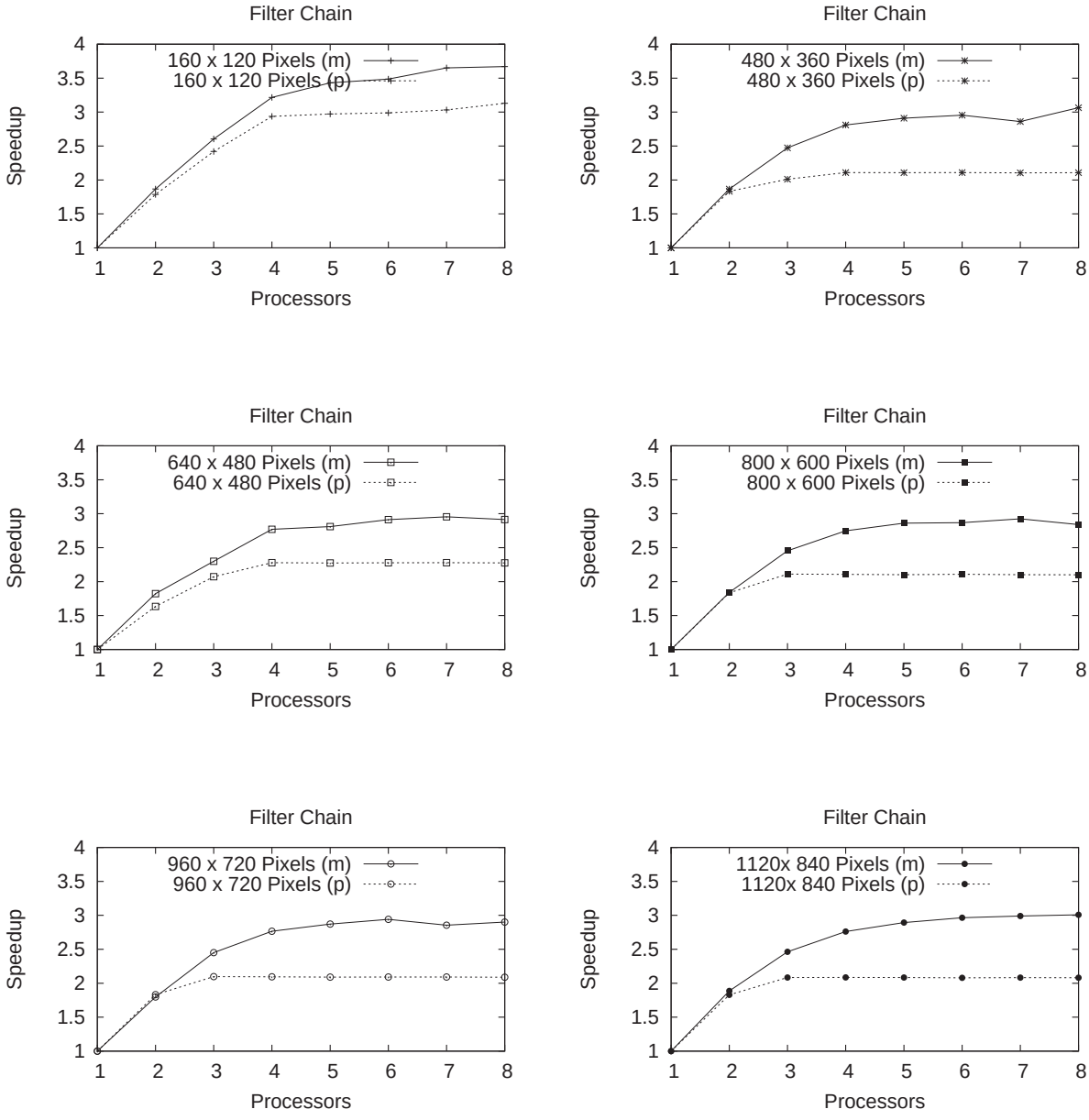


Figure 8: Filter chain measurement vs. prediction