2009 International Conference on Computational Science and Engineering

High Performance Printing: Increasing Personalized Documents Rendering through PPML Jobs Profiling and Scheduling

Thiago Nunes*, Mateus Raeder*, Mariana Kolberg*,
Luiz Gustavo Fernandes*, Alexis Cabeda† and Fabio Giannetti‡

*FACIN, PPGCC, PUCRS
Av. Ipiranga, 6681 Pr. 32 - Porto Alegre, Brazil
Email: thiago.nunes@pucrs.br, mateus.raeder@pucrs.br,
mariana.kolberg@pucrs.br, luiz.fernandes@pucrs.br

†HP Brazil R & DA
Av. Ipiranga, 6681 Pr. 95C - Porto Alegre, Brazil
Email: alexis.cabeda@hp.com

‡HP Laboratories
1501 Page Mill Rd. MS1160 - Palo Alto, USA
Email: fabio.giannetti@hp.com

***Abstract*—The creation of personalized documents has become a common practice on digital printing area in the past few years. In order to deal with the growing demand, Print Shops Providers (PSPs) need to be able to print a large amount of documents in a short period of time. New languages to describe documents in uncomposed formats establishing the need for a pre-processing composition step (also called rendering step). This new step has introduced an additional computational cost for the whole printing process, what has become a critical issue on printing environments based on recent digital presses. In this scenario, the use of high performance strategies is an option to overcome those issues. Previous works introduced the use of parallel FOPs (Formatting Objects Processor) rendering engines to speed up the pre-processing of a single document. The current work describes a new, scalable and flexible approach to increase the throughput of the pre-processing phase and defines a consistent metric to estimate the computational cost to render a PPML document. With those advances, the authors point out as future work alternatives to develop a parallel solution, capable of scheduling the document jobs increasing the overall throughput.**

***Keywords*-profiling documents; high performance printing; variable data printing;**

I. Introduction

Nowadays several companies are specialized on printing a large quantity of high quality personalized documents on a per recipient base. In order to customize the creation and design of documents, the use of Variable Data Printing (VDP) has became an established technique.

In this context, there is a need for an efficient language to deal with VDP. Personalized Print Markup Language (PPML) [2] was created for this purpose by Print On Demand initiative (PODi) [10], which is a consortium of companies that leads the digital printing market. In this study, the PPML Template (PPMLT) is used. The main difference between PPML and PPMLT is that in the first one there is a set of documents with all variable data already merged in, whereas in the second one, there is only a template of the document and data can be added separately. Templates are composed by static and variable portions and may be designed along with VDP techniques. The static parts are the ones that are not modified and are composed, for instance, by images, document, and page layout. The variable parts are merged using a database, generating many different instances of the template. All generated document instances are encapsulated on a single file, which can be referred as a job. Within jobs, the eXtensible Stylesheet Language - Formatting Objects (XSL-FO) [5] is applied to format the content inside the variable areas.

Currently, most of the presses are not able to interpret these job descriptions and formatting languages, thus several processes become necessary in order to print jobs correctly. One of these processes is the rendering phase, which represents a high computational cost [3]. This work introduces a new parallel approach for improving the throughput of the VDP documents during the pre-processing phase and establishes a new metric to evaluate the input jobs, considering their characteristics. The application of this new metric will determine *a priori* the computational cost to render a document job, making it possible to apply scheduling strategies to identify better performing printing jobs queue configuration.

This article describes on Section II an overview about the importance of the document rendering process, and introduces some basic concepts about the subject. Section III describes the functionalities and characteristics of the implementation, of the tool responsible for the rendering process, and it also relates the objective of this study, presenting the motivation for metrics to evaluate the input files. Section IV encloses testing

978-0-7695-3823-5/09 $26.00 © 2009 IEEE
DOI 10.1109/CSE.2009.203

aspects, such as the environment used to execute the solution (Section IV-A) and the application input test cases (Section IV-B). Section V presents the results obtained throughout this study, and, finally Section VI describes conclusions and points future works.

II. Scenario Overview

This section presents the current printing workflow scenario and issues. Print shops controls a job queue, with several document sets to be printed. This requires high processing capability since a Press can print in excess of a page per second. Furthermore, print shops usually employ several presses in parallel in order to increase the jobs consumption bringing the overall throughput well above the one page per second mark.

Since the printers can not natively interpret non Page Description Languages (PDL), a document composition stage must be applied before the printing phase. Therefore, it should be converted to a Page Description Language (PDL) format, such as Portable Document Format (PDF) [9], Post Script (PS) [11], Scalable Vector Graphics (SVG) [13] or others. Once this process is finished, the job can be sent to the printer.

Before describing the processing environment, it is relevant to highlight the most important aspects of these XML (eXtensible Markup Language) [15] formats: PPML and XSL-FO. PPML is a standard language for digital print built from XML developed by PODi which is a consortium of leading companies in digital printing. PPML has been designed to improve the rendering process of the content of documents using traditional printing languages. Thus, it introduces the reusable content concept where data, which is used on many pages, can be sent to the printer once and accessed as many times as needed from the printer's memory. PPML is a hierarchical language that contains documents, pages and objects (which are classified as re-usable or as disposable). Unfortunately, PPML is lacking of strategies to give different importance to the objects since the rendering process can have significant differences and impact on the overall time to generate the final document. Some work has been already presented [1] in order to address this issue, which currently remains open and is outside the scope of this paper.

The variable data is merged and formatted inside the PPML object element using XSL-FO. The containing PPML element is named "copy-hole", which is a defined area in the PPML code that can contain the variable data expressed in XSL-FO, or a non variable content such as images. XSL-FO (also abbreviated as FO - from Formatting Objects) is a W3C [14] standard introduced to format XML content for paginated media. It ideally works in conjunction with XSL-T (eXtensible Styleshee Language - Transformations) [16] to map XML content into formatted content. Once the XSL-FO is completed with the formatted content, its rendering engine executes the composition step to lay out the content inside the pages and obtain the final composed document. The rendering process in our environment is carried out by Apache's rendering engine named FOP (Formatting Objects Processor) [4].

In order to transform XSL-FO into PDL format, it is necessary to extract it from the PPML or PPMLT job. The Composition-Tool (CTool) is responsible for this work and uses Simple API for XML (SAX) to filter XSL-FO and direct them to Formatting Objects Processor (FOP) [4]. FOP will render XSL-FO and give the result back to the CTool. This process is executed for all the XSL-FO documents in the job and the final output is a PDL ready to be printed out. Figure 1 illustrates this process.

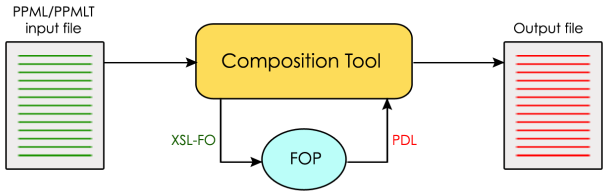


Fig. 1. Composition Tool Role

The computation done by the CTool is typically centralized on a single processing unit, resulting in a potential bottleneck in feeding all the presses during the rendering phase.

III. Proposed Approach

As mentioned before, pre-processing jobs must be completed in a timely matter. The problem is that the Digital Presses working in parallel produce a very high throughput and the CTool application should run at a speed comparable to the combined press speed, this is usually referred in the industry as "at engine speed". Therefore, the CTool represents a bottleneck on the entire process. This section presents two important steps to allow the creation of a smart parallel tool, capable of setting an *ad-hoc* configuration, that aims to improve the overall throughput of the composition stage and, consequently, the jobs processing.

A. Improved Parallel Solution

In previous studies [3], [8] strategies to improve the performance of the rendering process were introduced. These strategies were implemented adopting parallelism at the single XSL-FO level. This means that there was a single XSL-FO document parser and parallel components running were the composition engines, in our case FOP. The larger are the jobs, the heavier it becomes for the parser to process the XSL-FO document. This compromises the entire system performance in terms of scalability. From a certain size of the input document, the parallel approach didn't show any benefits compared to the sequential approach.

To deal with this situation we have identified a **new parallelization strategy**. Since the main issue on the previous strategies was the centralized document parser, not only the FOP engines were replicated, but also the parsing and extracting of XSL-FO documents improving the whole rendering process. Figure 2 illustrates this new approach.

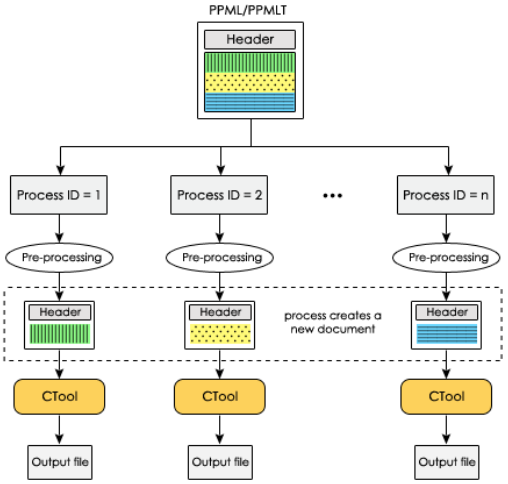


Fig. 2. New Parallel Approach

Each process will execute the CTool in parallel, receiving the same input file (PPML or PPMLT). The division of the work is based on the process *id* and it is based on the information stored in the record structure available in PPML documents. This information is necessary for the document to be separated correctly. It is important to mention that the number of processors is not predefined and the granularity of the resulting chunks can never be less than 1 record per process. Each process parses a single different chunk of the given document. These chunks are composed by several instances of the jobs documents. However, the initial input contains headers and other indispensable information that need to be on all the job portions executed by the processes. For this reason, each process creates a new job that just deals with its chunk and the headers indispensable information. The newly assembled job is now the input file for the CTool which performs the rendering process. At the end, an output file is generated by each one of the processes. Moreover, these files do not need to be re-joined together, since they can be printed independently.

The reviewer point is: how does the document know how many processors there are to be separated correctly? Now, this is possible due to the record structure available in PPML, I think we should stress this and also add that the number of processors is not predefined and that the granularity of the resulting chunks (number of records) and it can never be less than 1 record per process

B. Jobs Computational Cost Metric

It is important to consider that the rendering process is entirely focused on the XSL-FO documents. Therefore, just these elements should be considered in a PPMLT job. It is necessary to develop a consistent metric to predict the behavior of the rendering process. Using this approach, it will be possible to specify a weight for each document defining a better scheduling strategy for the parallel application to consume their jobs.

As mentioned before, during the rendering process, the FOP engine is responsible for transforming the raw XSL-FO component into a PDL object formatted as the pagination model. In most of the cases, the content of a XSL-FO is purely text, therefore most of the engines task is to position the text within the document, assembling the paragraphs and phrases inside their reserved area. In such scenario, if the amount of text embedded in a XSL-FO increases, so will the engines effort to complete the components rendering. Consequently, the **numbers of words** within the XSL-FO objects has been selected as metric to evaluate the rendering effort required for a given document. Experiments to support this choice are presented on Section V.

IV. Testing Environment

In this section the testing environment used to carry out the experiments is presented. First, the parallel architecture is introduced, then the input jobs used to evaluate and test the new approach are described.

A. Parallel Architecture

In high performance computing, a major concern is the choice of the parallel machine that better fits the application. In order to make the best choice, the characteristics of the implementation must be analyzed, for instance the amount of communication. The parallel application described in this paper does not need to perform any kind of communication, because all the processes have access to the initial input file and can discover which chunk of the job they will compute, based entirely on their ids. Therefore, the use of clusters seems encouraging.

Cluster of blades, which is a growing tendency, were chosen to execute and validate the parallel implementation. This architecture is composed by 10 blades with a quad-core Opteron 2.4 GHz processor and 8GB RAM memory each. This blades are interconnected through a 1Gb network.

On this kind of architecture, the MPI (Message Passing Interface) [12] is the most used standard to implement parallel applications and MPICH (MPI Chameleon) [6] is a stable library to provide MPI functionalities. Since the original application was developed on Java programming language, mpiJava 1.2.5 [7] with MPICH version 1.2.7 was the natural choice.

B. Input Jobs

As mentioned before, the input for the CTool application is a PPML or PPMLT file. This study only considers test cases on PPMLT format, because it is a more interesting format to companies that need a high flexibility to its marketing products. Each document of these files is mainly composed by pages, images, XSL-FOs and a determined number of words to be rendered. The characteristics of each test case are shown in Table I. All test cases represent real market situations and the layout of each one of them can be classified as a certain

TABLE I
DOCUMENT CHARACTERISTICS.

Test Case	Documents	Pages	Images	XSL-FOs	Words	Layout
Signage10000	10,000	1	5	7	100,000	Poster
Signage20000	20,000	1	5	7	200,000	Poster
Letter10000	10,000	2	5	8	1,850,000	Letter
Letter20000	20,000	2	5	8	3,700,000	Letter
Postcard10000	10,000	2	18	8	3,580,000	Postcard
Postcard20000	20,000	2	18	8	7,160,000	Postcard
Brochure10000	10,000	12	38	14	7,850,000	Brochure
Brochure20000	20,000	12	38	14	15,700,000	Brochure

type of document, for instance: poster, letter, postcard and brochure.

The first test case is called **Signage10000** and its documents are composed by only 1 page each, with 5 images and 7 XSL-FO portions. This file contains 10,000 documents, totalizing a content of 100,000 words within the XSL-FO components to be rendered, with approximately 10 words per document. This input file is typified by a poster like layout.

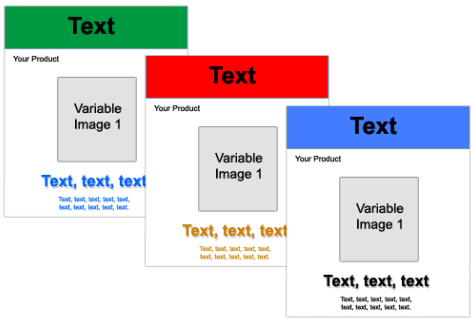


Fig. 3. Poster layout

The second test case, **Signage20000**, is very similar to the first one, containing 20,000 documents instead of 10,000. This file totalizes 200,000 words to be rendered, also presenting 10 words per document. Figure 3 illustrates three rendered documents of the first and second test cases.

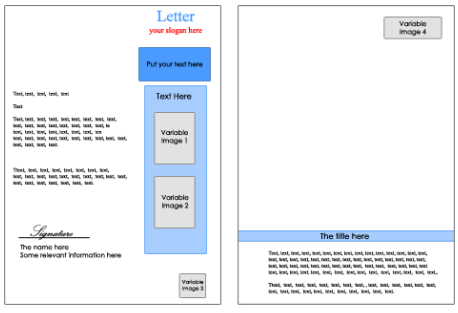


Fig. 4. Letter layout

The third input PPMLT is the **Letter10000**. This file is composed by 10,000 document, with 2 pages, 5 images and a total of 8 XSL-FOs each. A single document has around 185 words, totalizing 1,850,000 words. This file is typified by a letter layout.

Analogous to the third test case, the fourth one (**Letter20000**) contains 20,000 documents and 3,700,000 words to be rendered. Figure 4 presents an instance of a rendered document, corresponding to the Letter10000 and Letter20000 test cases.

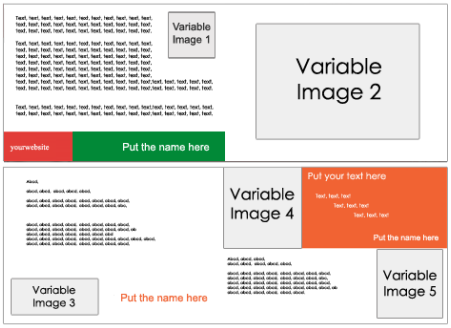


Fig. 5. Postcard layout

The **Postcard10000** file (fifth test case) presents around 3,580,000 words (with approximately 358 words per document), being composed by 10,000 documents. Individually, the documents have a postcard layout, including 2 pages, 18 images and 8 XSL-FOs.

A variation of this test case, is the file called **Postcard20000** (sixth test case). This test case possesses 20,000 documents, with the same configuration of the Postcard10000, summing up to 7,160,000 words. Figure 5 demonstrates an instance of a rendered document, related to the fifth and sixth test cases.

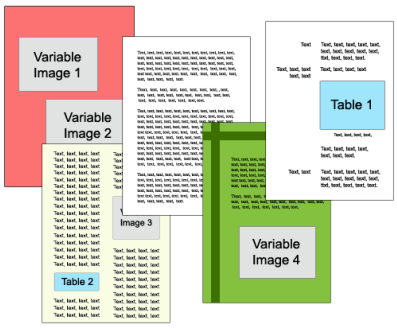


Fig. 6. Brochure layout

Finally, the **Brochure10000** (seventh test case) and the **Brochure20000** (eighth test case) presents 10,000 and 20,000 documents respectively. Their documents are composed by 12 pages, 38 images and 14 XSL-FOs. The **Brochure10000** test case has 7,850,000 words, while the **Brochure20000** has 15,700,000 words to be rendered, both with around 785 words per document and typified by a brochure layout. Figure 6 shows some pages of these two test cases.

TABLE II
SEQUENTIAL EXECUTION TIME

Test Case	Execution Time (seconds)
Signage10000	456.48
Signage20000	902.80
Letter10000	1,380.78
Letter20000	2,762.15
Postcard10000	2,449.70
Postcard20000	4,818.81
Brochure10000	4,875.73
Brochure20000	9,478.31

V. EXPERIMENTAL RESULTS

In order to evaluate whether the chosen metric is representative, some experiments over the previously presented input files presented were carried out. The experiments were performed using both: the sequential and the parallel version of the tool. Through the analysis of the sequential version, it was possible to check out if the chosen metric represents correctly the computational cost spent on the execution of each test case. The parallel version was used to estimate the gain that could be obtained through the job profiling in order to execute several jobs in parallel.

A. Metric Validation

The first step was to evaluate a sequential execution and record the times. The sequential approach constitutes our reference point to evaluate the proposed metric. Table II presents the obtained execution times. Analyzing the table, it is possible to assert that the metric is a consistent parameter to predict the PPMLT processing effort.

Brochure10000 and Brochure20000 (the test cases that contain the greatest number of words) were the ones that took the longest time to be rendered: 4,875.73 and 9,478.31 seconds respectively. An interesting aspect can be observed: for each pair of documents with the same characteristics, when the number of documents duplicates the rendering time follows the same pattern. Furthermore, it is possible to see that the chosen metric allows the prediction of the computational cost of different test cases, even with several distinct characteristics. For instance, the situation of the Postcard20000 and Brochure10000 input files in which the processing time is similar, the number of words to be rendered is similar. On the other hand, the Letter20000 test case execution time is higher than the Postcard10000 file, since the second one has less words than the first one. The previous statements indicate that the metric is consistent and valid choice to measure the rendering phase computational cost.

B. Parallel Execution

The eight test cases were executed through the usage of the new parallel proposed approach. Considering the fact that each node is composed by a quad-core processor (Section IV-A), each available core was considered as a single processing unit. Thus, it is possible to execute the solution with up to 40 processes using only 10 quad-core nodes, in order to analyze the application scalability. This choice can be easily applied, since there is no effective communication among the processes that may cause any load balancing issues.

For each test case, 20 executions were carried out (from which the highest and lowest times were not considered) and the final time was given by the mean of the remaining executions. Following the obtained execution times, job profile analysis and speed-ups are presented.

Execution Time

In order to analyze the performance benefits presented by the parallel version and mainly to provide parameters to verify the advantages that would emerge from a smart scheduling strategy, the proposed solution has been executed, varying from 2 to 40 processes. Figure 7 illustrates the execution time graphics, with a table showing some of these values, both corresponding to the first test cases: Signage10000 and Signage20000.

Examining the graphics it is possible to notice that the obtained execution times get as low as 27.24 seconds for the Signage10000 PPMLT and 36.51 seconds for the Signage20000 PPMLT (both with 40 processes), representing an important improvement when compared to the sequential version execution time (around 429.23 and 866.29 seconds respectively).

Following with the tests carried out, the execution time graphic and the table on Figure 8) refer to the Letter10000 and the Letter20000 test cases. With these PPMLTs the execution time has reached a total of 73.31 seconds (Letter10000) and of 133.16 seconds (Letter20000), employing 40 processes. These times demonstrate a gain of 1,307.46 and 2,628.99 seconds, respectively, when compared to the sequential tool.

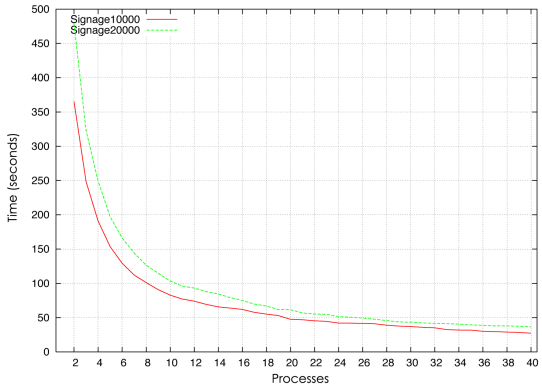
Analyzing Figure 9, which presents the execution time of the Postcard10000 and Postcard20000 test cases, it is possible to see that there was a time reduction of 2,334.46 seconds (for the first one) and 4,587.86 seconds (for the second one) with the usage of 40 processes.

Finally, Figure 10 illustrates the execution time for the test cases: Brochure10000 and Brochure20000. For the Brochure10000 test case (rendering time of 275.01 seconds) the reduced time gets up to 4,600.72 seconds and 9,084.42 seconds on the execution time of the Brochure20000 (rendering time of 393.88), both with 40 processes.

Job Scheduling Through Profiling

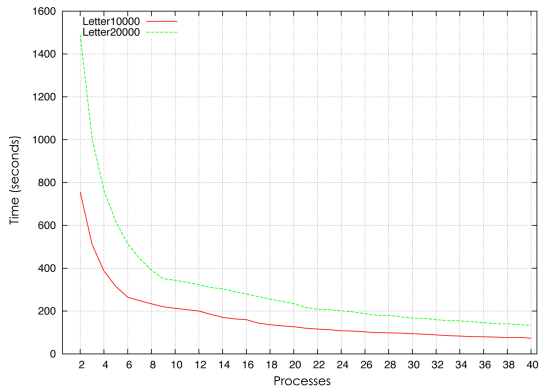
In order to verify the resource allocation benefits obtained from jobs profiling and scheduling, the metrics previously introduced were applied to two distinct scenarios. These scenarios were proposed contrasting the serialization of the jobs computation with the usage of a single instance of the proposed parallel version and the execution of several jobs in parallel, with several instances in parallel.

Taking in consideration the execution of the Brochure20000 test case with 30 and 40 nodes, it can be observed that the obtained gain gets up to 117 seconds. On the other hand, it is possible to execute the Brochure20000 with only 30 nodes



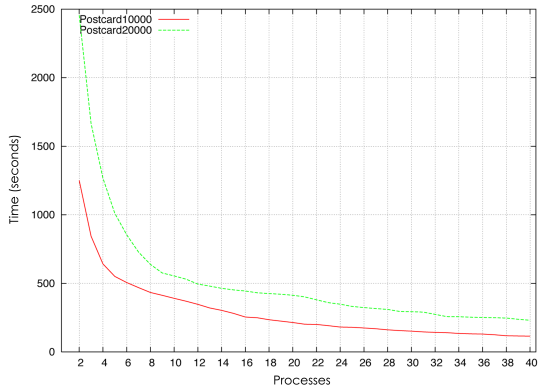
	Processes								
Test Case	2	5	10	15	20	25	30	35	40
Signage10000	365,4280	153,6790	82,7848	63,9060	47,5197	42,0170	36,9885	31,7585	27,2462
Signage20000	479,3860	197,192	103,4100	79,1275	61,5102	50,6657	43,3004	39,4415	36,5156

Fig. 7. Execution time - Signage files



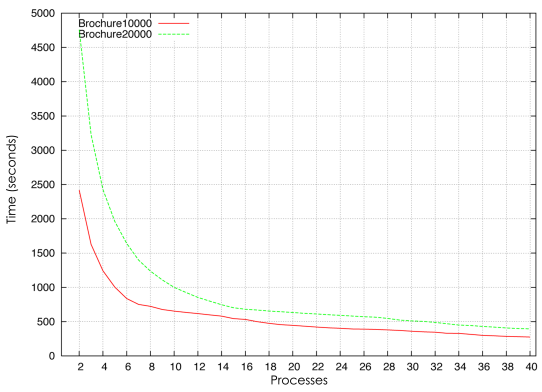
	Processes								
Test Case	2	5	10	15	20	25	30	35	40
Letter10000	755,2760	315,1770	212,7580	163,5190	126,7910	106,7380	94,2263	80,7686	73,3160
Letter20000	1490,4700	1005,9600	343,9520	291,2490	234,3730	196,0830	167,0520	150,8870	133,1620

Fig. 8. Execution time - Letter files



	Processes								
Test Case	2	5	10	15	20	25	30	35	40
Postcard10000	1250,4200	550,7620	390,9160	281,1860	214,5950	179,5120	151,5270	132,0750	115,2380
Postcard20000	2455,5600	1011,3200	554,0580	452,7600	413,1570	331,9250	293,3120	253,1960	230,9570

Fig. 9. Execution time - Postcard files



	Processes								
Test Case	2	5	10	15	20	25	30	35	40
Brochure10000	2418,1400	1003,6900	653,1690	544,8820	445,5760	393,1940	360,3260	314,9720	275,0120
Brochure20000	4758,4500	3228,3500	1000,1300	703,0570	633,2942	581,7620	510,6790	444,1165	393,8859

Fig. 10. Execution time - Brochure files

and allocate the other 10 to another job, like, for instance, Postcard20000. This way, it would be possible to execute both test cases in just around 554 seconds, which corresponds to the rendering time of the slowest one on the mentioned configuration. If these jobs were serialized in a way that all the resources were allocated for one job and than to the other, the total execution time would be 623 seconds, meaning 79 seconds more. At first, this number may not seem significant, but it corresponds to approximately 15% of performance gain. Another situation of performance gain, if it was compared to the jobs serialization, is the example of executing Letter10000, Postcard20000 and Brochure10000 files with respectively 5, 15 and 20 nodes. Through the parallel execution of these jobs, there would be a total time of 452 seconds instead of the 578 seconds obtained with the serial execution employing all the available resources. In this scenario, the time reduction gets up to 126 seconds, which represents a significant 22% performance improvement.

Speed-up

In order to evaluate the new approach scalability, on this section speed-up graphics are presented, corresponding to the eight test cases previously described. Figure 11 illustrates the obtained acceleration.

Through the analysis of the speed-up graphic it can be seen that the acceleration factor remained between 15 and 25, demonstrating a great benefit if compared with the parallel versions proposed on previous works. The Signage10000 file was the one which exhibited the lowest speed-up among all the test cases, but, on the other hand, the Signage20000 PPMLT showed the greatest speed-up, along with the Brochure20000 input file. The other PPMLTs maintained an acceleration factor around 20. Despite the fact that the obtained speed-ups do not represent a situation very close to the ideal, it is possible to notice that the obtained gain is substantial and do not reach a deceleration phase even with 40 processes. This way, it can be

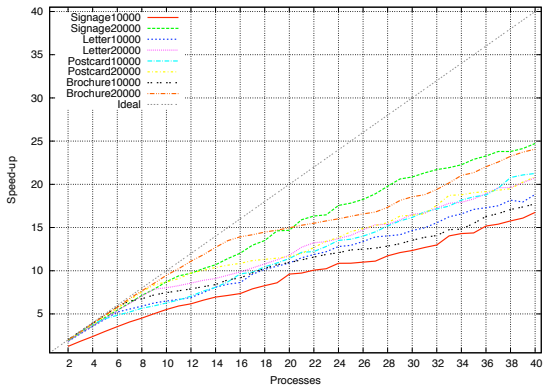


Fig. 11. *Speedup* for the eight test cases presented

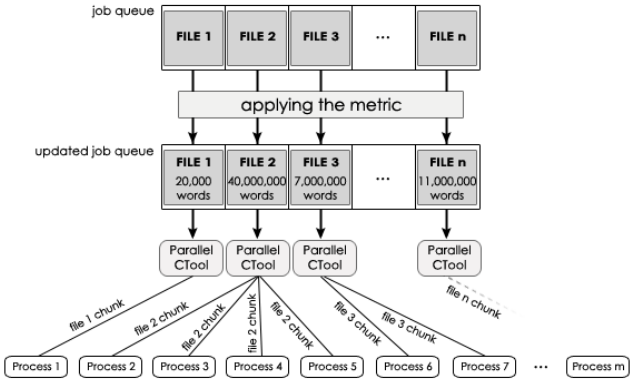


Fig. 12. Smarter Parallel Approach

seen that the new proposed approach is scalable, presenting a significant performance gain when more processes are added.

Considering that there is no communication among the processes, it was expected that all the speed-ups would be closer to the ideal line. The speed-up distance, from the ideal line, increases from a certain number of processes. This occurs because when the grain (number of documents to be generated per process) decreases, the initialization time (which involves reading and breaking of the original PPMLT file) becomes expensive, compromising the parallel application performance. Therefore, the faster is the execution the more significant will be the initialization time, reducing the speed-up factor.

VI. Conclusion and Future Work

Results show that a major improvement was obtained using the new parallel strategy, surpassing previous achievements in [3], [8]. Furthermore, this study demonstrates that the proposed metric (number of words within a XSL-FO) could be validated and also that to predict the document's behavior pattern. Based on this analysis, print shops may re-organize the order of job processing queue to acquire the highest throughput possible.

There are still new features and strategies that may contribute to increase the throughput of the application and its performance. Future research could be to use the application itself to schedule the documents, improving the overall throughput. In this context, there would be a preemptive queue of PPML or PPMLT files and throughout the analysis of the documents (in execution time), the application would decide which is the best way to organize the queue. For this situation it is really important to know the best way to assign a priority for each document, based on its processing weight. Therefore, the metric defined in this work would also address the queue management needs.

Another future work would be for the application to decide how many processing resources are necessary to render a single PPMLT (or PPML) job, having several jobs being executed concurrently. However, the solution described in this work, only processes a single document at a time. Thus, it would be necessary to develop a top-level application that, by the use of the metric, would determine the best quantity of resources needed for a given job and for the execution of the parallel CTool. Figure 12, illustrates the described strategy.

Considering the potential research lines and according to the outcome of this work, a parallel version which could adapt itself to many different situations points to future work.

References

[1] D. D. Bosschere. Book ticket files & imposition templates for variable data printing fundamentals for PPML. In *Proceedings of the XML Europe 2000*, Paris, France, 2000. International Digital Enterprise Alliance.

[2] P. Davis and D. deBronkart. PPML (Personalized Print Markup Language). In *Proceedings of the XML Europe 2000*, Paris, France, 2000. International Digital Enterprise Alliance.

[3] L. Fernandes, F. Giannetti, R. Timmers, T. Nunes, M. Raeder, and M. Castro. High Performance XSL-FO Rendering for Variable Data Printing. In *SAC'06: Proceedings of the 21st ACM Symposium on Applied Computing*, pages 811–817, Dijon, France, 2006. ACM Press.

[4] FOP. Formatting Objects Processor. Extracted from http://xml.apache.org/fop/ at March 21st, 2008.

[5] W. E. Kimber. Using XSL Formatting Objects for production-quality internationalized document printing. In *Proceedings of the XML Europe 2003*, pages 1–20, London, UK, 2003. International Digital Enterprise Alliance.

[6] MPICH. The MPICH Home Page. Extracted from http://www-unix.mcs.anl.gov/mpi/mpich1/ at March 23rd, 2008.

[7] mpiJava. The mpiJava Home Page. Extracted from http://www.hpjava.org/mpiJava.html at March 23rd, 2008.

[8] T. Nunes, L. G. Fernandes, F. Giannetti, A. Cabeda, M. Raeder, and G. Bedin. An Improved Parallel XSL-FO Rendering for Personalized Documents. In *Euro PVM/MPI'07: Proceedings of the 14th European PVM/MPI Users Group Meeting. Recent Advances in Parallel Virtual Machine and Message Passing Interface (LNCS)*, volume 4757, pages 56–63, Paris, France, 2007. Springer.

[9] PDF. Adobe PDF. Extracted from http://www.adobe.com/br/products/acrobat/ at March 28th, 2008.

[10] PODi. Print on Demand Initiative. Extracted from http://www.podi.org/ at March 29th, 2008.

[11] PS. Adobe - Adobe PostScript 3. Extracted from http://www.adobe.com/products/postscript/ at March 28th, 2008.

[12] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: the complete reference*. MIT Press, 1996.

[13] SVG. Scalable Vector Graphics (SVG). Extracted from http://www.w3.org/Graphics/SVG/ at March 28th, 2008.

[14] W3C. The World Wide Web Consortium. Extracted from http://www.w3.org/ at March 29th, 2008.

[15] XML. Extensible Markup Language (XML). Extracted from http://www.w3.org/XML at March 21st, 2008.

[16] XSL-T. XSL-Transformations. Extracted from http://www.w3.org/TR/1999/REC-xslt-19991116, section References, at March 29th, 2008.