IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-6, Issue-6, Jun 2018 E-ISSN: 2347-2693

Stack improving optimization feed with multi core task display interface

Sumalatha Aradhya[1*], N.K. Srinath[2]

[1]Computer Science and Engineering, R V College of Engineering, Mysuru Road, Bengaluru, Karnataka, India
[2]Department of Computer Science and Engineering, R V College of Engineering, Mysuru Road, Bengaluru, Karnataka, India.

[*]*sumalatha.aradhya@cedlabs.in, Tel.: +91 9900609254*

Available online at: www.ijcseonline.org

Accepted: 21/Jun/2018, Published: 30/Jun/2018

***Abstract*—** The real time embedded software development requires expertise for developing critical software. The safety critical embedded development has major concerns as the final target code should execute with less size and more speed. The increased stack size and reduced execution speed lowers the performance of embedded software. In the paper, a method to overcome the rapid growth of the stack is proposed and multicore load balancing issues are addressed. The model for stack improving optimization feed and multi core task balancing display interface is derived in the paper. A unique approach with the method of providing optimization hints during the development phase through interactive display interface is suggested in the paper. The experiment is conducted by considering a concave set of functions with a task dependency derivation cost. The best execution result being obtained by using stack, improving optimization feed interface.

***Keywords*—**Embedded software, interactive, display interface, multicore task balancing, parallelism, programmer, optimization, task dependency

I. INTRODUCTION

The scarcity of advanced programming tools and safety critical environments enforces programmer to rewrite sequential programs into parallel programs [1]. The details of the optimization are abstracted with 20% of the code is optimized as per Eigenmann et al. [2]. The code optimization through compiler optimization is not opted in safety critical systems due to the criticality of embedded software requirements. This is because the compiler induces extraneous code and this leads to deviation in the result. This triggers the requirement of the programmer not only with parallel coding proficiency, but also with code optimization skill set. To overcome the issues, the proposed solution is to provide an interactive display interface to the programmer suggesting possible optimization corrections to the code. The proposed interactive interface provides further scope of task balancing across multiple cores while programming.

Hall et al. [3] mentioned that the existing embedded compilers do not provide the features of designated initializations. Hence, code automation through scripting languages or through manual effort is a difficult task. Pennycook et al. [4] stated that cores were kept serialized frequently due to resource sharing conflict and communication overheads. The existing method to resolve these issues is through the thread-checker tools and thread profile tools. Intel's Thread Building Blocks [5], Terra [6], valgrind [7] and gprof [8] are to name a few existing thread checker and profiling tools. Thread checker and profiler tool helps to improve the correctness of threaded applications. However, the tools induce complexities in the hybrid architecture environment. The complexities occur due to inefficient task distribution and improper utilization of processors as mentioned by Kaur et.al [9]. A task scheduling simulator with multiple scheduling algorithms is suggested by Vasiliu et al. [10]. Such optimization is analytically too complex. Hence, simulation approach is a new requirement of the critical embedded applications as the next era belongs to code modernization. The related work of the proposed model is discussed in section II. The mathematical model to derive the stack optimization and task balancing logic is discussed in section III. A. The working principles and analysis of the derivation model is discussed in section III.B. The section IV covers the experimental setup and discussion of results obtained.

II. RELATED WORK

Chmaj et al. [11] and Ró˙zycki et al. [12] proposed optimization of task balancing across cores. But, the core wise task distribution logic had low data processing and transmission rates. The Mathwork's Simulink model [13] has the continuous simulation, although it is impossible over a digital hardware. Modelica tool proposed by Fritzson et al. [14] addresses the simulation of industrial system applications and automations. But, Modelica tool is not recommended for measuring the interactive process and it

needs many external libraries [15]. Seidewitz [16] proposed an Abstract Language Framework (ALF) tool that has the model with more complex, stricter and more behavioural semantics, but cannot be expressed using graphical notation. Colored Petri as proposed by Roy et al., [17] were partly used, though are better than Simulink as they have clear and formal semantics. The Feedbags [18] is an application that generates a pass-through code. Nevertheless, ReSharper and Feedbags applications require class, interface R# and lack interactive feedback for task balancing. Eclipse [19] is widely used in the embedded world, but lacks interactive suggestions to the programmer.

SWIFT is a loop interactive feedback driven framework [20]. This framework acts as a plug in compiler for any setup. However, the users need to have a detailed knowledge of internal representation of the optimizer. Java's application tool by name PMD [21] and IntelliJ IDEA tool [22] provides stack analysis of the code, but lacks interaction with the IDE for core load balancing. The TenAsys's InTime [23] is an interactive tool and determines the exact sequence and precise timing of real time code execution. However, the tool does not show the core balancing statistics at the front end. The Irisa's Salto tool [24] determines the exact sequence and precise timing of real time code execution. The Salto tool also suggests the interactive assembly logics to the programmer. But, Salto tool does not show the graphical representation of the core loads. All the tools mentioned, namely PMD [21], IntelliJ IDEA [22], Swarm [20], InTime [23], and Salto [24] do not suggest the stack improving optimization logics to the programmer while coding. These drawbacks provided motivation to design interface with interactive displays.

III. METHODOLOGY

The embedded programmer should design and implement the logic with respect to specific system requirements and functional requirements. To help the programmer, an interactive display interface has been proposed in the paper and its logics are discussed next.

A. Derivation Model

The core wise task balancing is NP-hard (Non-Deterministic Polynomial) optimization problem[9]. Effective load balancing is needed to improve the performance and keep the system in stable condition [25]. The existing optimization logic for parallel computation of tasks is the generalization of partitioning problem using an LPT algorithm (Longest Processing Time) as proposed by Abdullah et al. [26]. The derivation logic is represented by using following system model terms.

- n:Total number of processors
- γ: The symbol represents the list with task dependencies.
- $Proc_i$: The processor available to take the load and i represent processors in the list and the value ranges from 0 to n-1 where each i represents a single core
- σ: Existence of similar function set core wise
- $Func_{iexists}$: No of function set mapped with similar task dependencies
- $Func_{jexists}$: Function set to be swapped with similar task dependencies
- S: Capacity size of the processor
- MinAvailCores: Amount of contiguous resource available to load function mapped

Referring to look ahead selective sampling algorithm proposed by Abdullah et al. (2016) [26], sampling set of dependency list is obtained and formula for obtaining task dependency list γ is represented in Eq. (1). The set of cores available with a similar set of functions is sampled at the interval 0.25. The sample rate of 0.25 is proven as the minimum interval for load balancing algorithm by Falco et al [27]. The probability of occurrence of a function to be swapped with function in given software is an exponential growth. The ratio of available processor and the existence of swappable function provide the set of the dependent tasks. Thus, with the minimal available set of cores Proci with function data mapped to tasks, Funciexists, the task dependency list γ is derived as follows [26]:

$$\gamma\left(Func_{iexists}\right) = 0.25e^{-\left(proc_i \div \sigma\right)} \quad (1)$$

Where, σ represent occurrence of function and derived by considering processor available with probability of delta changes from its previous state.

A linearity check is done between the function set exists between successor processors available in task distribution list. That is, the function to be mapped in one core is compared with another core available next. The cores are mutually exclusive if the tasks cannot occur simultaneously. That is, the tasks are independent and distinct. A linear dependency equation for identical function occurrence is derived as shown in Eq. (2).

$$Func_{iexists} + \left((-1)Func_{jexits}\right) = 0 \quad (2)$$

Consider two sets of cores where one set of cores represents a function to be swapped and another set of cores with the function to be mapped. The function to be swapped and mapped is checked across n and m set of processors. The function is swapped by considering the size of the memory available at that core. The formula for distribution logic represented in Eq. (3) [27,28]

$$\sum_{i=1}^{n} (Proc_i \times Func_{jexists}) \div n = \text{MinAvailCores} \quad (3)$$

The constraint for minimal set $Proc_i$ requires that each task j in function $Func_{jexists}$ is assigned to at least one core in $Proc_i$ list. Therefore, balancing j task for $Func_{jexists}$ must satisfy the constraint that the sum of cores in the $Proc_i$ list must not exceed the capacity size S. From Eq. (3), it is clear that the distribution of function task that exists in any core lies within the range of 1 to a maximum capacity size, S. Thus, derive minimal available set of cores, *MinAvailCores* as shown in Eq. (4).

$$\sum_{i=1,\, j=1}^{n,m-1} Func_{jexist} \times Proc_i \le S \quad (4)$$

There has to be at least one minimum resource available at MinAvailCores list to schedule independent function tasks. The availability of resources is arbitrary constant and when allocated is represented by using the value 1. Hence, the default value of the dependency cost is considered as 1 when the task depends on other core's task, otherwise is set to 0.5 when shifted. Once the sampling set of independent and dependent function tasks across cores are derived, final task distribution cost of the tasks is derived.

B. Working Principles And Analysis Of The Derivation Model

The task path shown in Fig.1 emphasizes cores with pool of tasks. Few tasks exists in the task pool are completed only after the completion of other tasks in other core's task pool. This induces dependency cost across the cores and increases the waiting time of the task before executing next task in the pool. The core's task dependency cost is reduced when the task executing in another core's task pool is added to the current task pool. Such linear dependency of tasks across cores is checked and moved to or from the existing task pools by using Eqs. (3) and (1) respectively. For further illustration, consider example shown in Fig. 2. Core, C1 has task pool with 9 tasks. Core C2 has task pool with 15 tasks. Core C3 has task pool with 11 tasks and Core C4 has task pool with 13 tasks as Fig.1

By applying eq. (2), a linear check of tasks is performed. The task that is independent in its parent core, but linear dependent on task in other core can be added to the mapping list. The mapped tasks can be added to new task pool of cores available in MinAvailCore set. The MinAvailCore set, that is, minimum sets of cores ready to be balanced are identified by applying Eq. (4).

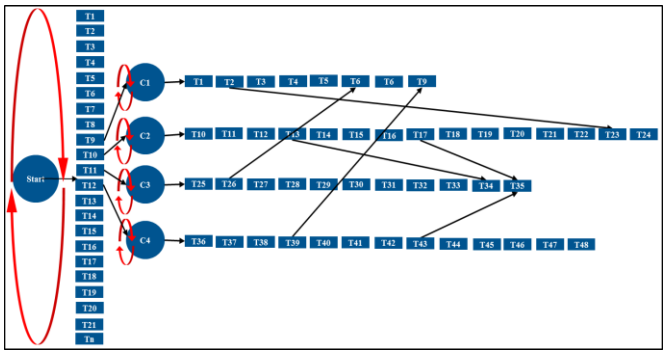

Figure 1. Existing task pool across multicore before optimization

After applying distribution logic, the optimized task pool at multicore are represented as shown in Fig. 2. Figure 2 represents new task pool and newly added tasks are shown in red colour.

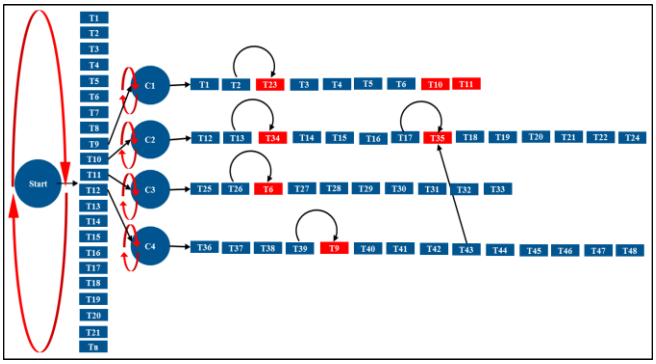

Figure 2. Existing task pool across multicore before optimization

The task balancing logic is derived through four cases.

- Case1: T2-T23 balancing at core C1. Tasks T6 and T9 exempted from core C1. Tasks T23, T10 and T11 added newly to the core C1. Tasks T10 and T11 are added as they are not linearly dependent on any tasks. This extra addition reduces the load of core C2
- Case 2: T13-T34, T17-T35 balancing at core C2. Task T23 is exempted from core C2. Tasks T34 and T35 are added newly to the core C2. Through Eq. (3), it is found that the capacity of core C2 was exceeding. Both predecessor and successor cores are compared by referring to the task distribution list obtained using Eqs. (1) and (2). The independent tasks that available next to execute in the task distribution list are considered for such cases. The minimal sets of cores available for loading are derived by using Eq. (4). The core with least load considered for loading extra tasks
- Case 3: T26-T6 balancing at core C3. Tasks T34 and T35 are exempted from core C3. Task T6 is added

newly to the core C3. Task T35 was needed by task T17 at core C2 and by task T43 at core C3. Based on the derivation logic discussed, the task is added to the task pool of core C2. That is, by mapping the task set using Eqs. (1) and (3).

- Case 4: T29-T9 balancing at core C4. None of the tasks are exempted from core C4 and task T9 is added newly to the core C4.

The analysis of the multi core task balancing similar to above cases along with task dependency cost matrix are shown in Figs.3 and 4

Task Dependency Cost Matrix –with no core load balancing

Core 1	Task Cost	Dependency Cost	Core 2	Task Cost	Dependency Cost	Core 3	Task Cost	Dependency Cost	Core 4	Task Cost	Dependency Cost
T1	1		T12	1		T25	1		T36	1	
T2	1	1	T13	1	1	T26	1	1	T37	1	
T3	1		T14	1		T27	1		T38	1	1
T4	1		T15	1		T28	1		T39	1	
T5	1		T16	1		T29	1		T40	1	
T6	1		T17	1		T30	1		T41	1	
T7	1		T18	1		T31	1		T42	1	1
T8	1		T19	1		T32	1		T43	1	
T9	1		T20	1		T33	1		T44	1	
T10	1		T21	1		T34	1		T45	1	
T11	1		T22	1		T35	1		T46	1	
T12	1		T23	1					T47	1	
T14	1	1	T24	1					T48	1	
T15	1										
Core 1 Load	16		Core 2 Load	14		Core 3 Load	12		Core 4 Load	15	

Figure 3. Task Depencency Cost Matrix –with no core load balancing

In Figure 3, four cores with existing task pool are shown. With the mutual exclusion of tasks by cores, independent tasks are shifted to corresponding core. The optimized core task load balancing analysis is shown in Fig. 4.

In Figure 4, the tasks are balanced across cores. The tasks shifted are available next to the corresponding linear dependent task to the pool. When task dependency cost across cores gets reduced, an effective task balancing is achieved. Figure 5 represents the best approximation solution achieved through core load balancing. The vertical axis shows the task dependency cost and horizontal axis shows the cores in the list. The red bar represents reduced task dependency cost core wise and the blue bar represents a task dependency cost without core load balancing. This approximation solution is provided to the display interface library.

Task Dependency Cost Matrix –with core load balancing

Core 1	Task Cost	Dependency Cost	Core 2	Task Cost	Dependency Cost	Core 3	Task Cost	Dependency Cost	Core 4	Task Cost	Dependency Cost
T1	1		T12	1		T25	1		T37	1	
T2	1	0.5	T13	1		T26	1		T38	1	1
T23	1		T34	1		T9	0.5		T39	1	
T3	1		T14	1		T27	1		T40	1	
T4	1		T15	1		T28	1		T41	1	
T5	1		T16	1		T29	1		T42	1	1
T6	1		T17	1		T30	1		T43	1	
T7	1		T35	1		T31	1		T44	1	
T8	1		T18	1		T32	1		T45	1	
T10	1		T19	1		T33	1		T46	1	
T11	1		T20	1		T36	1		T47	1	
T22	1		T21	1		T24	1		T48	1	
Core 1 Load	12.5		Core 2 Load	12		Core 3 Load	12.5		Core 4 Load	14	

Figure 4. Task Depencency Cost Matrix –with core load balancing

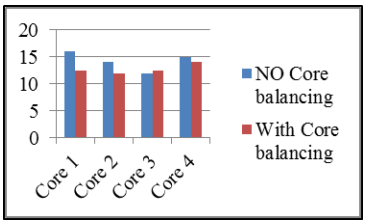

Figure 5. Best Approximation Solution across cores

IV. RESULTS AND DISCUSSION

For testing or research purpose, single core ARM processor from ARDUINO UNO board is considered and environment test set up as shown in Fig.7. In the experiment, single core functional balancing is used to prove the concept that the multi core load balancing achieved is as similar to balancing the single core. In the experiment by referring to the single core task balancing, simple concept of toggling GPIO pins is used. The GPIO pins are toggled with a delay of 10 microseconds by enabling and disabling the GPIO through nested function calls and without nested function calls.

The timing analysis of the scenario with a nested function call is captured in CRO snapshots as depicted in Fig. 7. Similarly, the timing analysis of the scenario without a nested function call is captured in CRO snapshots as depicted in Fig. 8. The scenario produced with nested function calls resulting in rapid stack growth attains the execution time of 3.120 ms as shown in Fig. 7.

Figure 6. Test Envoirment setup

The scenario produced without nested function calls resulting in execution time of 1.000 ms as shown in Fig. 8. The compiler does not optimize the code because in every calling function few tasks are running. Hence, by the time required function is called, an extra time of 2.120 ms is elapsed. Overall, debugging the cause of 2.120ms delay in real time critical system impacts the time and cost of the programmer

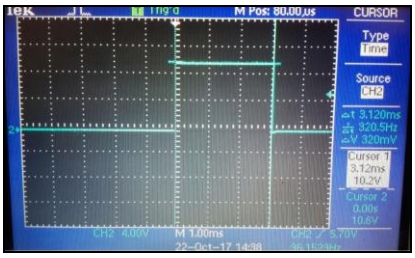

Figure 7. Senario with nested function calls- increasing in rapid stack

Resolving time related issues at the stage of code development avoids extensive core overloading and is evident from the result shown in Figs. 7 and 8.

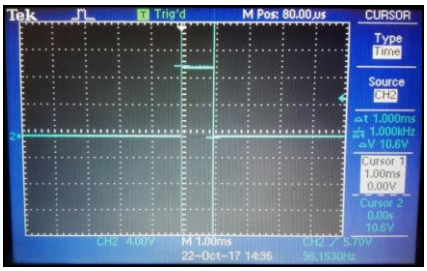

Figure 8. Senario with nested calls- decrease in rapid stack growth

The call stack depth becomes less and returns from the stack frame with unwanted memory references are skipped. Such optimal solution when given through the interactive suggestions and core load chart as shown in Fig. 9, the programmer can write an optimized code while coding. The Figure 9 shows the core load chart to hint programmer to set task affinity with further scope of optimizing the load across cores. The bar chart with no core balancing displays 13% task load at the core. The bar chart with 5% core load balancing displays 5% of task load at the core. It is evident from the result that with the reduction of call stack depth and with the appropriate task load distribution at the cores, the task load on the core is reduced by 8%. The overall task execution time at the target is optimized due to optimal task load at the core. Hence, the best approximation solution is achieved. The interactive display interface acts as an optimization, carrier at the front end and at every phase completion of the code. The interactive IDE generates the report with graphical representation while the development is in progress.

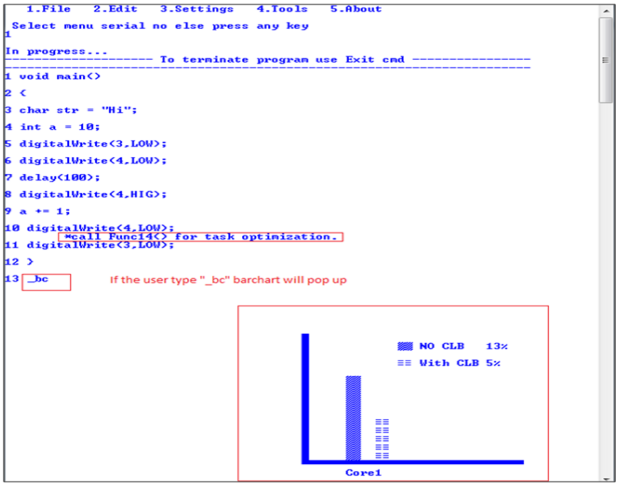

Figure 9. Intractive display interface to the programmer

V. CONCLUSION AND FUTURE SCOPE

A unique idea to help programmer through interactive display interface is proposed in the paper. The interface provides statistics of task balancing of multi cores during the development stage. The objective to achieve best execution speed is attained by using a derivation model of stack improving optimization feed interface. The stack growth is reduced by optimizing the task balancing logic across cores and experiments are conducted by considering function calls in use. The multi core task balancing derivation logic discussed in the paper provides the best approximation solution to obtain decreased stack size and increased execution time. The interactive display interface provides further scope of code optimization logic along with core wise load chart representation of the display. The multicore task balancing interface hints the programmer to allocate the cores with balanced loads. By referring to the interactive display interface, programmers can write optimized code while coding. Thus, this paper contributes successfully for the future embedded programming world. The next scope of work is to derive worst case task distribution logic for the stack improving optimization feed.

VI. REFERENCES

[1] Bahl, A. K.; Baltzer, O.; Rau-Chaplin, A.; and Varghese, B. (2012). Parallel Simulations for Analysing Portfolios of Catastrophic Event Risk. Workshop Proceedings of the International Conference of High Performance Computing, Networking, Storage and Analysis (SC12).

[2] Eigenmann, Rudol.; Hoeflinger, Jay.; and Padua, David, (1998). On the Automatic Parallelization of the Perfect Benchmarks. IEEE Transactions on Parallel and Distributed Systems, volume 9, number 1, January 1998, pages 5-23.

[3] Hall, Mary; Padua, David.; and Pingali, Keshav. (2009). Compiler Research: The Next 50 Years, FEBRUARY 2009 VOL. 52, COMMUNICATIONS OF THE ACM, pp. 60-67,

[4] Pennycook, S.J.; Hughes, C.J..; Smelyanskiy, M.; and A, S. (2013). Exploring SIMD for Molecular Dynamics Using Intel R Xeon Processors and Intel R Xeon PhiTMCoprocessors , Parallel Computing Lab, Intel Corporation, IEEE publication, 2013

[5] Contreas, G., and Martonosi, M. (2008) Characterizing and improving the performance of Intel threading building blocks. In 4th International Symposium on Workload Characterization (IISWC 2008), Seattle, Washington, USA, September 14- 16, 2008 (2008), pp. 57–66

[6] DeVito, Zachary.; Hegarty, James.; Aiken, Alex.; Hanrahan, Pat.; and Vitek, Jan. (2013). Terra: A Multi-Stage Language for High Performance Computing PLDI13 June 16-22, 2013, ACM

[7] Nethercote, Nicholas; Seward, Julian. (2007). "Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation". Proceedings of ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI 2007).

[8] Graham, Susan L.; Kessler, Peter B.; and Mckusick, Marshall K.(1982). gprof: a Call Graph Execution Profiler, Proceedings of the SIGPLAN '82 Symposium on Compiler Construction, SIGPLAN Notices, Vol. 17, No 6, pp. 120-126

[9] Sapinderjit Kaur, Kirandeep. Kaur, Amit.Chhabra, Parallel Job Scheduling Using Grey Wolf Optimization Algorithm for Heterogeneous Multi-Cluster Environment, International Journal of Computer Sciences and Engineering (IJCSE), Volume 5, Issue 10, E-ISSN: 2347-2693, Oct 2017, pp 44-53.

[10] Vasiliu, Laura.; Pop, Florin.; Negru, Catalin.; and Mocanu, Mariana, (2017). A hybrid scheduler for many task computing in big data systems. Int. J. Appl. Math. Comput. Sci., 2017, Vol. 27, No. 2, m`, 385–399

[11] Chmaj, G.; Walkowiak, K.; Tarnawski, M.; and Kucharzak, M.(2012). Heuristic algorithms for optimization of task allocation and result distribution in peer-to-peer computing systems, International Journal of Applied Mathematics and Computer Science 22(3): 733–748,

[12] Ró˙zycki, R.; Waligóra, G.; and Weglarz, J.(2016). Scheduling preemptable jobs on identical processors under varying availability of an additional continuous resource, International Journal of Applied Mathematics and Computer Science 26(3): 693−706,

[13] Mathwork's Simulink Tool R2017b (2017). Generate C and C++ code optimized for embedded systems, MathWorks.

[14] Fritzson, Peter.; Bachmann, Bernhard.; Moudgalya, Kannan,; Casella, Francesco.; Lie, Bernt.; Kofranek, Jiri,.; Haumer, Anton,.; Geusen, Christoph.Nytsch.; and Vanfretti, Luigi (2017). Introduction to Modelica with Examples in Modeling Technology, and Applications, Modelica Publication, 2017

[15] Frenkel, Jens.; Schubert, Christian.; Kunze, Günter.; Fritzson, Peter .; Sjölund , Martin.; and Pop, Adrian (2011).Towards a Benchmark Suite for Modelica Compilers: Large Models, 8th International Modelica Conference - Dresden, Germany - 20-22 March 2011, ISBN: 978-91-7393-096-3

[16] Seidewitz, Papyrus.Ed (2015). Tool Paper: Combining Alf and UML in Modeling Tools – An Example with Model Driven Solutions, A specification by CEA, LIST.

[17] Roy, Nilabja.; Dabholkar, Akshay.; Hamm, Nathan.; Dowdy, Larry.; and Schmidt, Douglas (2008). Modeling Software Contention Using Colored Petri Nets, MASCOTS 2008, Baltimore, MD, USA

[18] Amann, S.; Proksch, S.; and Nadi, S (2016). FeedBaG: An Interaction Tracker for Visual Studio, In Proceedings of the 24th International Conference on Program Comprehension Tool Track, 2016

[19] Vogel, Lars (2013). Eclipse IDE: Java programming, debugging, unit testing, task management and Git version control with Eclipse (3rd ed.). Leipzig: Vogella. ISBN 978-3943747041.

[20] Gonnet, Pedro.; Schallery, Matthieu.; Theunsyz, Tom.; and Chalk Aidan B. G. (2013). SWIFT - Fast algorithms for multi resolution SPH on multi core architectures, 8th international SPHERIC workshop Trondheim, Norway, June 4-6, 2013

[21] Rutar, Nick.; Almazan, Christian. B.; and Foster, Jeffrey.S (2004), "A Comparison of Bug Finding Tools for Java". ISSRE '04 Proceedings of the 15th International Symposium on Software Reliability Engineering, IEEE

[22] Saunders, Stephen; Fields, Duane K.; Belayev, Eugene (March 1, 2006), IntelliJ IDEA in Action (1st ed.), Manning, p. 450, ISBN 1-932394-44-3

[23] Neumann, Dean.; Kulkarni, Dileep.; Kunze, Aaron.; Rogers, Gerald.; Verplanke, Edwin. (August 2006). Intel Virtualization Technology in Embedded and Communications Infrastructure Applications, Intel Technology Journal of application of virtualization to embedded systems.

[24] Bodin, Francois.; Chamski, Zbigniew.; Eisenbeis, Christine.; Rohou, Erven.; and Seznec, Andre. (1997). Salto: GCDS, A Compiler Strategy for Trading Code Size Against Performance in Embedded Applications, systems Pro jet CAPS Publication internet ,1153 ,December 1997, from https://hal.archives-ouvertes.fr/inria-00073718/document

[25] Deepali Simaiya, Raj Kumar Paul, "Review of Various Performcae Evaluation Issues and Efficient Load Balancing for Cloud Computing" IJSRCSEIT, Mar-Apr;3(3) :pp. 943-951, 2018

[26] Abdullah, Loai.; and Shinshoni, Lian. (2016). Look Ahead Selective Sampling for Incomplete Data, International Journal of Applied Mathematics and Computer Science., 2016, Vol. 26, No. 4, 871–884

[27] Falco, I. De.; Laskowski, E.; Olejnik, R.; Scafuri1, U.; Tarantino, E.; Tudruj, M. (2013). Load Balancing in Distributed Applications Based on Extremal Optimization, Applications of Evolutionary Computation, Springer Verlag (Ed.) (2013) pp. 52-61

[28] Korte, Bernhard.; and Vygen, Jens.(2006). Bin-Packing, Combinatorial Optimization: Theory and Algorithms, Algorithms and Combinatorics 21, Springer, pp.426–441, ISBN 978-3-540-25684-7

Authors Profile

Mrs. Sumalatha Aradhya received B E degree from Dr. Ambedkar Institute of Technology, Bangalore in year 2000 and M Tech from M V J College of Engineering, Bangalore in year 2006 and currently perceives her PhD in the field of parallel computing from R V College of Engineering, Bangalore. The author has 16+ years of experience with diversified exposures to telecom, avionics and memory computing areas across the industries such as IntelliNet Technologies, Bangalore; L & T Infotech, Bangalore; HCL Technologies, Bangalore and Quest Global, Bangalore. Her research interests are compilers, high performance computing, and embedded systems

DR.N K Srinath is currently working as Dean of Academics at R V College of Engineering, Bangalore. He has 33 plus year of experience in teaching and his area of research are systems engineering and operations research. He has more than 52 international journal publications and he is author of several text books related to microprocessors and data base systems. He served as advisory committee member for various national and international proceedings, and was part of expert committee member of UGC as chairman and is active member of several education bodies' advisory committees.

