Session 2: Low Level Analysis

Session chair: Stefan M. Petters (University of York, UK)

Presentations

A discussion of the influence of scratchpad memories on the WCET analysis has been subject of the first presentation given by Lars Wehmeyer. The authors came to the conclusion, that the WCET can benefit considerably from scratchpad memories, with no extra effort on the analysis.

Jürgen Stohr presented a method to control the influence of PCI DMA transfers on the WCET. Focus of this work is a PC style architecture and how to make that more suitable for real-time applications.

Synergetic effects of cache related preemption delays were presented by Jan Staschulat. This centers around multiple executions of the same straight line code as it is used in the automotive industry and how that can be analyzed with much better results than previous simplifying approaches.

The last talk by Oleg Parshin described a method to analyze instruction caches for individual components and how the results for those components can than be integrated into an overall WCET approach when the components are composed to form systems.

Discussion

The following discussion centered on the feasibility of componentization of real-time systems and the impact on WCET analysis.

Peter Puschner: (To Oleg Parshin) In this analysis you say you analyze all procedures and this means even the first step of the analysis is for a concrete memory architecture. So is there a step which abstracts away from the concrete layout and size of cache?

Oleg Parshin: The cache line size needs to be known.

Peter Puschner: Did you try to store the actual memory references in some more abstract way? For example a procedure is used in several different contexts with different memory layouts and by storing the information in a more abstract way, you could reuse information gathered in previous analysis steps.

Oleg Parshin: We used the AbsInt tool, hence had no influence.

Stefan M. Petters: But technically it should be possible to have two analysis steps, use abstract information about memory regions used and then come up with the final result in a computational stage instantiating the analysis results.

Oleg Parshin: You need to know concrete size of the cache.

Stefan M. Petters: That's when you abstract away from the relative location and just say, even if you fix the cache size you can just say, within the module I'm doing the about that and when I use it in some context I need to know where it's actually located, but that should be more troublesome.
On a more general scale we have heard about scratchpad memories twice today. Does anybody know about the industrial uptake of those, compared to locked caches, which is the obvious alternative?

Christian Ferdinand: In most recent developments you have the choice: You have some amount of memory which may be either used as scratchpad memory, or cache and this may be configured at boot up, or even split it some this and some that way.

Stefan M. Petters: But is it really used?

Christian Ferdinand: In automotive it is truly used, because they start of with a design with an expected life span (in terms of delivery) of 6 years, but after 3 years they get so many requests for changes, that whatever it possible with that piece of hardware will be done. But not automatically, as they currently don't know how to use this.

Iain Bate: Where you use abstract interpretation in aiT style tool for cache analysis, one thing we have found working with industry industrial use of poll status true/false for runtime exceptions they use abstract interpretation. In practical code they get a lot of *mays*, which need either manual inspection or a safe estimate. While this is a very different problem, when you are using this actually for cache analysis, what percentage of memory accesses do you get as *mays*, because that can be seen as wasted resource and one would try to cut that down.

Christian Ferdinand: The problem is this is a really good question that comes up all the time The problem is that really depends on the application. One can write application code, where this prediction would be extremely poor….

Iain Bate: I'm talking about well written code.

Christian Ferdinand: For example, we as a company only provide the tools, but have not much insight into the application, but there is one example from Airbus, where they applied it on what they call real-time benchmark, which is basically an old version of the fly-by-wire code of the 340. This is generated with the SAO Scade tool mainly and we had a look at the Coldfire 5307 with 8 KB. One problem is no-one really knows what the real WCET is, but what they did is a lot of measurements and they also used a legacy method based on measurements and a lot of argumentation what could be the WCET. Our result was between the results of the legacy method and the measurements. They believe they had about 20% overestimation with the legacy method and if we assume this correct, we get about 10% overestimation.

Iain Bate: I'm not talking about the overestimation and the overall execution time analysis results. It's actually how many memory references you get as mays. Say you have got a million memory references, what percentage?

Christian Ferdinand: I do not know precisely where the overestimation comes from. Could be pipeline analysis or non-precision of the cache analysis. It's a factor of 8 between a cache hit and a cache miss. If you assume all the 10% overestimation going into cache analysis, you come up with 1% or 2 % of wrongly predicted mays, so it's very precise, it's 99% precision in that specific case.

Stefan M. Petters (to Jan Staschulat): Setting aside future computational problem, how do you think you can get that to scale as it is now, in terms of looking for a branch. The examples you had were not too big and with the direct mapped caches or two way set associative caches you have been on the lower end of things. So how would you think you can manage the bigger problems?

Jan Staschulat: The high complexity is due to the analysis of multiple preemptions, because several cache states have to be analyzed in the control flow graph. To simplify one can abstract from specific cache states and merge several cache states. This will reduce the analysis complexity but also lower the precision. So the question is: *"how much precision can you get and how much effort you are willing to pay"?*

Peter Puschner (to Jan Staschulat): Is there any first others? Have you ever measured the time you needed to solve the problem?

Jan Staschulat: They are long, indeed. Several hours.

Peter Puschner: Even for your small examples?

Jan Staschulat: Yes. We are thinking about a different implementation. The analysis of several preemptions has to be simplified. Instead of considering all paths of a preempting process, several cache states should be merged leading to a reduced number of states. The underlying concept of data flow algorithm considers than less states and the analysis complexity is reduced.
But what is really necessary is the analysis of multiple process activations, because this is the loop of straight line programs of automotive applications. Since a cache is not useful for linear programs, a second program execution can be seen as a loop, and consequently cache lines are reused.

Alexander von Buelow (to Jan Staschulat): Does you approach also work for interrupts or are they not considered?

Jan Staschulat: At the moment interrupts are not considered. Interrupts can happen any time, as often as you wish. You have to assume it occurs n times in a given time period, and the user has to specify this information somehow. I don't know how to consider interrupts in this analysis yet. You might want to consider jitter also.

Stefan M. Petters (to Jan Staschulat): In your data regarding your multiple preemptions, you just compared your results to previous approaches, but didn't actually get to the point what the real simulation results were in context of an arbitrary number of runs of a program with a specification of the number of preemptions.

Jan Staschulat: I didn't show the simulation results in this table, but I ran some experiments with the ARM Developer Studio and I instrumented the debugger to stop the preempted process at a preemption point and run the preempting process and then let the preempted process continue. I did this for all combination of n given multiple preemption points. The comparison showed that our CRPD analysis was close to the simulated results. Of course this verification by simulation works only for small examples.

Jan Gustafsson (to Lars Wehmeyer): I have another question to this scratchpad idea. It's actually two questions: The first one is: *"how does your method scales with larger applications"?* and the

other question is: *"your analysis method is part of the compiler?"*. What are your opportunities to get this into a real compiler? Any thoughts about this or contacts? Because obviously this has to be done inside the compiler to work. I think the scratchpad idea is really great, but how is the future?

Lars Wehmeyer: Regarding the first part I guess your question aims at the direction of ILP and complexity issues. Right now we're using the simplex ILP solver and we are formulating our models accordingly as an ILP and then giving that to the solver. But we have been discussing this problem with other groups at our University, especially theoretical computer science, and we have come to the conclusion that what we describe are actually knapsack problems and they belong to the group of packaging problems. Usually you can find a linear time approximation and you can also find a polynomial time approximation. So there is the possibility by trading in some of the precision of the final solution for the execution time. Using these approach and thoughts for scaling it to larger benchmarks and larger applications should not be a big problem. So you have to find out how far you are of the optimal solution and how much computation for finding an allocation of objects to the memory are we willing to accept in order to get closer to a certain precision.
And the second question, right now the approach is integrated into our research compiler, but I guess, in order to really integrate it into any other compiler, all you would need is the information about the analysis that was performed, you would need either a static analysis or profiling and I think with that information and an energy model which models the environment and you know the energy required to access the different types of memory and you know about execution frequencies it should be actually possible to integrate it more or less into any compiler. Of course you would need to find some way of interfacing, but actually it is just a way of telling the linker where to place the different objects in memory. That's all you need to place a certain memory object onto the scratchpad. You just tell the linker, put this to that address and that address is mapped to scratchpad memory. So I guess, by cleverly interfacing knowledge you have within the compiler and having analysis regarding the runtime behavior of your application, externally solving the problem and then back annotating the information you get from solving the problem into your compiler, maybe it could even be done by solving the optimization problem and then using a linker script so it's not that deep within the compiler.

Iain Bate: I just wonder, how many of those working on scratchpad memory have had a look at the work SoC people have done on optimizing performance and power characteristics? People like Frank Vahid or Tony Givargis are still on this work. It's because they have done an awful amount of work in their domain. They have not combined it with execution time analysis, very much like testing systems in order to see what the worst case performance is but they have done a lot of work in how to configure your memories, caches and scratchpads for the best effect. I wonder whether anybody has looked at that communities work.

No one!

Peter Puschner: I would like to make a comment on this work of modularized or reusable WCET analysis: I think this is really an interesting piece of work and I would like to encourage you to push this further, because I think this is really one of the open questions on WCET analysis: How to reuse information for timing analysis? You mentioned the software development process and what we all do is, we compile modules and we don't compile all modules again and again, but we compile only the modified ones and link them with the others. I think this should also be true for WCET analysis. If we analyze a piece of code, we should be able to store this information in a way that can be used later on and make the analysis easier. So you don't want to do the whole complex analysis again and again, but rather do the complex analysis tasks only once and then

store the information in a way, so we can reuse our results cheaply, so we don't have to go through all the analysis and all the modules, functions and libraries again and again. And I think this is one step in that direction.

Stefan M. Petters: Another step in that direction, which is upcoming work looking at real-time components on the vendor side. A real-time operating system, where you don't have the actual hardware it's running on, so you want to provide as much information in terms of timing as you can and then instantiate that on a particular piece of hardware or a particular environment and I think that is certainly worthwhile.

Guillem Bernat: Can we have a feel of what is the impact of these recompilations of the WCET? From your experiments, what is the difference in the execution time of a module when you execute it somewhere else? Is it for example, a 1% or a 50% difference?

Oleg Parshin: I can't tell you exact numbers.

Guillem Bernat: When you compile a module and recompile it you expect the functional behavior to be the same. The answer to Peter is: You can do that if you assume that if you add some small module somewhere else does not change that much and therefore you are sort off in the same dimension, then you can, in all other cases you have to do a full analysis end to end. The question is the ability to do it component-wise is only true if the components have a meaning and they don't really depend too much on the context they are running in.

Peter Puschner: Yes, in some way. Of course, if you design a system it depends on the interfaces. In this way, the degree in which you can decompose the problem depends on the interference of the different parts and of course, if the interference is too large, it might not be worthwhile, but then the question is whether the solution is the right solution. We can't recompile and reanalyze all this. If this takes an hour, you would not want to do it every time you change a module. If it really takes an hour, then the solution is probably not the right solution: Isn't the architecture wrong?

Jan Gustafsson: One way of dealing with this could be to analyze parts, whether they are context sensitive or not and if they are not then you don't have to do this all over again. That could be a part of the analysis.

Stefan M. Petters: It's actually a question whether you want to go down the route and separate the context non sensitive from the context sensitive stuff analyze one and wait for the other to happen or whether you split the analysis rather than the application into bits you know and things you can't know at this stage and try to get the second stage as small and simple as possible, because you don't want to have engineer XYZ sitting somewhere and trying to figure out what you initially were trying to do.

Iain Bate (to Guillem Bernat): I've got a question for Guillem Bernat: the WCET analysis work presumably subject to regression testing and trying to carry all those many measurements as possible forward for a task once you do a change to that task that is important. Have you thought how you might handle that?

Guillem Bernat: Well, that's exactly why I asked the question. In measurement based analysis we assume that the impact on the execution time once you make changes to the code are actually quite small. So you can reuse a lot of measurements on your code, unless there are some pathological cases. The issue is, the approach we are actually using is you test your components

and carry that forward, but once you have your system finally assembled, you do a full analysis of the whole system once, assuming there is no more changes on the code.

Iain Bate: Because what happens in the functional testing world, they make unit tests only on the units which have changed which is isolated from the rest of the system and then they do very limited progressively less integration tests for the number of changes, so they do some scenarios of functional testing, so in the end this assumes that the changes don't ripple through too much.

Guillem Bernat: You don't even need to recompile the code, you just relink it with a slightly different link map and then all the addresses change and then there is allegedly nothing you can do. I think the point here is – going back to my point of different systems – in a absolute safety critical system, in order to get meaningful values, you will completely redo an end-to-end analysis, any time you introduce a small change. If you have a system with less criticality, like the automotive industry, then they accept there is an error on that value, but it's more or less in that value and the error is not an order of magnitude. From my experience they know that small changes in the code won't double the execution time. You get an extra couple of cache misses but that's what it costs.

Peter Puschner: I'm a bit surprised to hear this, because my guess would be: If you do some static analysis and also analysis of interferences as proposed, and you change a module, you would be able to see the effect of this change on the interference, because I see now I would use this and that cacheline too, which I didn't have before and therefore I see there is now an interference with module X. But if you measure, how would you be able to see this interference, because you don't go into the detail?

Guillem Bernat: You don't know where it is, but you observe the effect and that's the philosophy change, the change of mentality. A large project with a couple of 10.000 lines of code, the number of memory accesses made is in the order of millions, hundred millions. The people who design caches are very clever and they have been making for a long time something, which guesses the impact of the memory accesses. So yes, you change a module, hence something that was a cache miss becomes a cache hit, but the miss happens now somewhere else. Statistically speaking after a very large number of memory accesses this averages out. So we say: a good cache mechanism hits about 9x% would you expect that a change in the program makes your cache miss 50%?

Peter Puschner: But that takes us away from hard real-time, right?

Guillem Bernat: That's what I'm saying. There is an absolute guarantee somewhere else and as soon as you move over edge of the absolute safety critical then you are at this gate.

Iain Bate: An observation from a Federal Aviation Authority perspective. They are very concerned about the use of caches, the multimode sort of things like pipeline effects, because of these sort of problems and therefore a lot of work looks into what the issues are. A lot of systems are just avoiding caches wherever possible turning them off, etc. Some of them are using them, but they haven't got a grip on what to do. Things like CAS20, which is the guidance document, still funding more and more work in this area on how to handle this problem and not just from the execution time analysis perspective.

Peter Puschner: Just to clarify that: You don't observe the task you have modified in isolation, but again you are looking at the whole system. So you don't just make a local analysis after a

change, but you have to go back to your global analysis of the whole system to get out the knowledge of all the interferences, which is a more complex problem.

Guillem Bernat: Let me take a step back. We are doing measurement based WCET analysis as opposed to static analysis. Measurement based analysis does not address the issue of what happened to that particular cache line, but you just measure the execution time and if it happens to be longer, then you can assume that maybe there was an additional cache miss or something. Now when you move away from characterizing each individual memory access that's when the large numbers are on your side. That's the first piece. Secondly, when you do testing or when you analyze a small component, there are two steps to manage this. One is to determine how long it takes for a component to execute in isolation and the second step is in the high level analysis to put all things together. But those are completely separate processes. This is not linear, but N log (N) with the size of the program and is independent on the testing. Even if you do it manually, putting together all the information would not take longer then the time to compile your code. At least it's in the same order of magnitude. Deriving the information of each module is what takes time, but that is linear with the complexity of the module, so there is no computational explosion, but depends on how you do your testing. You could do 4 weeks of testing and would get much better data, but you want to separate the issues of deriving information on the module and how to put that information together. So that's where one comes up with the capability issue. In fact, one can even deal with the problem of a whole subsystem being not developed. Because you say "I assume that module X is going to take that much and that's a design estimate" and then can reason whether one is 300 times over budget or within budget. The key point here is, if you move away from the absolute, absolute guarantee that you get the absolute, absolute worst case life is much easier. Besides no safety critical system had that information.

Peter Puschner: And probably not as safe.

Guillem Bernat: All safety critical systems have the built in assumptions, that a chip will fail, data corrupted etc.

Iain Bate: The issue is showing that this is not a cause of failure, one can accept one part of a redundant system to fail, but you don't want all of them fail at the same time. Since in those systems, the redundant parts are mostly working in locked step, your assumption of having statistics on your side does not hold, because they are supposed to behave identical.

Peter Puschner: I think you are right that people assume that systems or parts of systems fail, but what they usually do is they try to avoid to introduce any sources of failures themselves. In order to reduce complexity, and one strategy to do that is to separate the system into components and try to make them as independent as possible, to keep the interfaces and interferences really small. What you are trying to do is, to develop one component and try to verify correctness including temporal correctness and then rely on what you got. If you then compose these components, you want to be sure they still work. You want to have that for timing not only for functional aspects. In the moment I start to neglect the interfaces and interferences, if you just measure, you avoid the view on the interference on purpose. You just neglect that.

Guillem Bernat: Not really I think that's a different thing. You measure module A and measure module B and then you put A and B together considering the worst possible interference and that's the kind of thing we do. The probabilistic analysis framework allows you to reason: "*What is the worst possible combination of this and this?"* Then you can actually ask whether this combination is feasible or whether you can reduce the pessimism.

Peter Puschner: If you want to argue about it. It has to be simple, otherwise it's too complex. You take two tasks and can hardly do it. If you just measure it's still a probabilistic statement, but you can't prove anything.

Stefan M. Petters: In the end, I think what drives all of it is cost. Assuming one can pour 200 million dollars into each individual plane to make it 100% save or one can be 100% - 10^{-20} sure and save that money, I see people being willing to go down the route of saving that money.

Guillem Bernat: We had an argument with someone saying: *"Well, worst case is too pessimistic for us! We want our testing, we don't believe it.*" Testing is too optimistic, the absolute worst case is too pessimistic and they want to have something in between. If it is good enough, if it is less likely to fail than any other component of the system, then that's good enough for them.

Peter Puschner: Let me refer to a different area where quite the opposite of what you claimed happens and that's when you look at communication protocols. For cars, where, for example, CAN seemed to work quite well for a very long time and now companies are getting into trouble and they start to see that what they have to use and they are all now getting more or less into the time triggered business, because they realize that building on probabilities gets them into trouble when complexity reaches a certain level. So now they go down and say: *"We can't do that anymore. We have to have systems, which are decomposable, where we have an isolated view on each of the components and where we reduce the interference between components really to a minimum"*.

Stefan M. Petters: But they have still a probabilistic argument for a component failure.

Peter Puschner: Of course, there are always hardware failures. But what they try to avoid is to build in probabilistic failures themselves, because then it's difficult to argue about dependencies of things happening.

Stefan M. Petters: But there's still the question of who has to foot the bill if something goes wrong.

Iain Bate: An interesting thing is, how big a guarantee do you want? And what that comes down to is cost of human life. When I have done enough assessment for a system in, let's say, avionics and then it comes down to issues, like how many people are on the aircraft, which countries is it flying over, because there are unfortunately certain differences, in what a nationals life is worth, and in follow up what is the cost of litigation in court. What they do is certainly a trade off between the costs of safety, because they come to the point where they are happy to write off these costs of litigation if they loose an aircraft. The question is: *"how far are we willing to go?"*, because there is no such thing as absolute guaranties. There are always probabilities, even in static analysis it's still probabilities. For example your analysis tool aiT. You are relying on the compiler your compiling your tool with, do you have formal proof of the compiler? No. Do you have formal proof of the processor model? No. You have to rely on the documentation and observations. So it all comes down to probabilities of cost of life.

Stefan M. Petters: But it's more complicated like that. Then you have authorities like the FAA, which just want to make sure, they can't be sued and don't have to foot the bill to pay for the development. If you have those agencies involved it's a bit more tricky.

Iain Bate: You can't sue the FAA. They don't hold responsibility for life, that's the thing of the aircraft companies and operators. The FAA has no legal responsibility. In the military world it's different though, because the MoD may be sued.

Guillem Bernat: In the end the question is: "How much is enough?" I can do a very rough testing on a simulator and that tells me that this code executes for 1000 cycles my deadline is a million cycles, that's good enough. You would consider that enough. The tools are not reliable, but nevertheless you are sure enough it does not blow up. If you need something which costs a lot of money and time, which does not improve that statement, then people just don't use it. If it's clearly schedulable, that's OK. Even if it's clearly un-schedulable, that's also OK. In the cases at the borderline is when you make the distinction between the application domains, whether simple end-to-end measurements are OK or whether you need a very elaborate method.