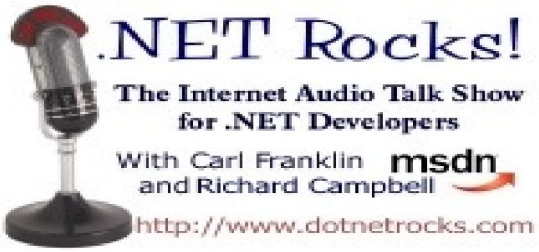

[HTTP://www.dotnetrocks.com](http://www.dotnetrocks.com)

Carl Franklin and Richard Campbell interview experts to bring you insights into .NET technology and the state of software development. More than just a dry interview show, we have fun! Original Music! Prizes! Check out what you've been missing!

Text Transcript of Show #671
(Transcription services provided by PWOP Productions)

Paul Bone Goes Parallel with Project Mercury
June 14, 2011
Our Sponsor

Lawrence Ryan: .NET Rocks! episode #671, with guest Paul Bone, recorded live Friday, June 3, 2011.

[Music]

Lawrence Ryan: This episode is brought to you by Telerik and by Franklins.Net - Training Developers to Work Smarter and now offering video training on Silverlight 4.0 with Billy Hollis and SharePoint 2010 with Sahil Malik, order online now at franklins.net. And now here are Carl and Richard.

Carl Franklin: Thank you very much and welcome back to .NET Rocks! It's your .NET podcast, the .NET podcast.

Richard Campbell: That's what we like.

Carl Franklin: That sound kind of vain, doesn't it? Because there's a lot of .NET podcasts.

Richard Campbell: There is a few.

Carl Franklin: We sound good though.

Richard Campbell: We do something right.

Carl Franklin: Something. Hey, man, what's up?

Richard Campbell: Not too much. You know, summer is finally showing up. It's getting warm around here. The barbeques have been dragged out. It's time to cook meat and drink beer.

Carl Franklin: Hey, did you see that cookbook from GrapeCity?

Richard Campbell: Yes, I did. Hey, we're in it.

Carl Franklin: I think they're selling us. So, here's what GrapeCity did. They got a bunch of .NET gurus together and asked them for recipes. Richard has two recipes in there, I've got one and also a lot of guests that have been on .NET Rocks! people that you know that you go see speak and all that kind of stuff so I think you can get them at grapecity.com, but they're a lot of fun and good food too. Wow, your paella was ridiculous.

Richard Campbell: It's a bit of work to make, but it's a good piece when you're done.

Carl Franklin: Hey, let's get started with Better Know a Framework.

Richard Campbell: I love it.

[Music]

Carl Franklin: Today we're not knowing a framework, we're knowing a project at CodePlex.

Richard Campbell: Ah, yeah.

Carl Franklin: I'm on a roll here, man.

Richard Campbell: You're liking it, are you?

Carl Franklin: mediacompanion.codeplex.com, Media Companion is the original free to use movie manager and organizer that offers full XBMC integration. I don't even know what that means.

Richard Campbell: Nice.

Carl Franklin: But it sounds important. Simply put, Media Companion offers the facility to gather information from the Internet and make this information available to you in an organized manner. The information collected includes such things as posters, backdrops, plot summary, actors and actor images, ratings, etc. So, what's cool about this is because it's source code, you can see the code to scrape movie information from IMDb, to search the Internet for this kind of data and it shows you where to get it and how.

Richard Campbell: Nice.

Carl Franklin: It's kind of cool, and also for TV shows and it's had, let's see, what have we got for downloads? In the last seven days, about a thousand downloads and eight ratings, almost five stars.

Richard Campbell: Nice, people like it.

Carl Franklin: Yeah, they like it. Richard, who's talking to us?

Richard Campbell: I've got an email here from Jeremy Huppatz and he says, "Gentlemen," and there is a stretch for you.

Carl Franklin: Yeah.

Richard Campbell: "I'm writing to you guys to thank you for your ongoing efforts to keep schlubs like me informed on what's going on in the .NET community and for being an inspiration to me at both a professional and personal level."

Carl Franklin: Whoa.

Richard Campbell: Nice.

Carl Franklin: That's a great name for a band, The Schlubs. I love it.

Richard Campbell: "In the last eight months, I've finally stepped away from being a full-time permanent salary man to hanging out a shingle and running my own consultancy. A big factor at making that decision was the inspiration I took away from show #368 in which Carl and Mark Dunn spoke to Steve Smith about making the jump from permanent employee to small business operator." That's a while back, that show. It was like 300 shows ago.

Carl Franklin: That was, yeah. I don't even think you were there.

Richard Campbell: I wasn't. That was one of the ones where I was away and you went with Mark and talked to Steve.

Carl Franklin: Yup.

Richard Campbell: "As a result of that move, I've developed a newfound sense of self-respect and in the next two weeks I will have successfully completed my first consulting project, an enterprise-grade customer management data warehouse built on SQL Server 2008 R2 including elements of SSIS, SSAS, and SSRS. Life is looking great and now I'm seriously looking at next steps. For me, that will be a move to split my time between working in IT to pay the bills and starting to move into a field I'm truly passionate about. Now that I'm in a house that offers me some elbow room, I'm able to make effective use of the home recording gear I've accumulated over the past 10 years..."

Carl Franklin: Oh, geez.

Richard Campbell: "By producing some of my own tunes, getting some bands through for recordings and completing a formal qualification as an audio engineer at SAE."

Carl Franklin: That's awesome.

Richard Campbell: "I'm also looking at closing the IT dev music loop by creating a .NET infrastructure-based software solution that handles assets and workflow management for Cakewalk Sonar X1 production solution and looking to see if I can integrate that into pro-tools as well."

Carl Franklin: Wow.

Richard Campbell: "Carl's career as an initially reluctant IT guy who finally moved back to his first love, making music with talented people, has been a major source of motivation for me in this area and I'm truly grateful that I discovered .NET Rocks! Now, send me a mug, damn it!"

Carl Franklin: Just for the record. I may have been a reluctant IT guy, but I was a fully exuberant developer. I'm totally engaged.

Richard Campbell: Yeah. The funny thing is it tickles your brain in the same way music does.

Carl Franklin: Totally.

Richard Campbell: That concept of mastery that nothing is ever perfect, it can only get better.

Carl Franklin: Yeah and the creative aspect, the abstract aspect of it. You know, I used to love putting together PCs and I still do. I still do. I still love going to Newegg and buying all the pieces and it all comes in a box and it's like Christmas.

Richard Campbell: Right.

Carl Franklin: You open it up and you spend a couple of hours and nowadays they just work, but man, back in the day there was some serious -- you remember when motherboards didn't fit in cases?

Richard Campbell: Right.

Carl Franklin: They were just like off by millimeters and the screws didn't fit? It's maddening, but standards being what they are, stuff just works now.

Richard Campbell: These things have evolved.

Carl Franklin: Yeah.

Richard Campbell: Well, Jeremy, thanks so much for your email. We are inspired back, man. I'm glad it's working for you. We'll send you a mug down to Australia.

Carl Franklin: Absolutely.

Richard Campbell: If you've got questions, concerns, ideas for shows, you can send us an email at dotnetrocks@franklins.net or post up on our fancy new website at dotnetrocks.com. We love comments on shows and heck, even the guests will comment back.

Carl Franklin: In a perfect world. Richard, I'm very excited because our guest today is none other than Paul Bone. Paul is a PhD student at the University of Melbourne in Australia. He works on Mercury, a purely declarative logic programming language. We'll find out what that is in a minute. His PhD thesis topic is the automatic parallelization of Mercury programs. A paper describing Paul's recent work is due to appear in Theory and Practice of Logic Programming. He will be presenting this paper at the

International Conference of Logic Programming to be held in Lexington, Kentucky in July. Paul is also, and you're going to love this, check out this, a visually impaired downhill skier. Yikes.

Richard Campbell: So you're crazy.

Carl Franklin: Yikes.

Paul Bone: Yeah. Well, my wife said that since she too stained to do a PhD herself so it's just the one, just me, so I'm crazy enough to take on a PhD and ski.

Carl Franklin: Man, I'm a skiing impaired downhill skier. That's what I am. I got Ski Baba, the little beginner's slope and then the next one I went to was like the jaws of death and I went down on my face the whole way. It was great. Anyway...

Paul Bone: If you're not falling down, you're not trying hard enough so that's good.

Carl Franklin: Well, Paul, my very first question is in all the documentations you draw a line in the difference between logic programming languages and imperative programming languages. So what's the difference?

Paul Bone: There are two necessarily opposites but they often are not associated together.

Carl Franklin: Okay.

Paul Bone: It's more the imperative and the declarative sort of opposites.

Carl Franklin: All right.

Paul Bone: Most of your listeners have probably been familiar with imperative programming. This is you tell the computer what to do and you usually tell it a sequence of instructions and then you might use something like any of the NLS. So say you will leave this condition and do that thing and so on and you write loops, for loops, and so on and that's called imperative programming. In declarative programming, the programmer doesn't tell the computer how to solve the problem, but it just tells them to tell the computer what the problem is and often those descriptions are recursive and so looping is done through recursion.

Carl Franklin: When people think of declarative languages probably in a .NET space, we think of XML, XAML. Those are the kinds of declarative things that we deal with all the time and they're looking more and more like languages just because by expressing something in this declarative language, something happens. So is this essentially what you're talking about? Just not action and logic flow, but actual... I mean, why is it called a logic programming language then?

Paul Bone: Okay. So the logic part of that ScriptLogic and declarative are often associated together. A logic programming language is one that's rather than built out of statements and expressions, it's built out of logical goals so when it executes it tries to prove these goals. For instance, a goal might be somebody else's grandparent and to set aside that goal somebody had you find the parent who is the child of the grandparent but the parent of first child.

Carl Franklin: Yeah, I get it.

Paul Bone: Yeah. So it's...

Carl Franklin: So you give it a series of givens and then things that it has to prove and it figures out how to prove it.

Paul Bone: Yeah. it's built out of what's called honed clauses so you see the thing that it can curve and then you say the things that that can be made up from. So whether there's a conjunction of other goals or a disjunction of other goals and you can include if they aren't losing things in there like that.

Carl Franklin: So what kinds of applications is this type of language, and Mercury in particular, really good for? What kinds of problems does it solve the best?

Paul Bone: So when people think of logic programming, they often think of prologue which I believe was originally developed to help people with natural language processing.

Richard Campbell: Yeah, that's a blast from the past. That's from the 1980s.

Carl Franklin: Way back in the 1980s.

Paul Bone: Yeah. So that's often what it's been used for in the past, but we've seen that logic programming is generally useful for writing any types of programs. Our Mercury compiler is written in Mercury, for instance.

Richard Campbell: How recursive of you.

Paul Bone: Thank you. This is called self-hosting when your language can compile its compiler.

Richard Campbell: Right.

Carl Franklin: Yeah.

Paul Bone: And Mercury has been self-hosting for I think 15 years now. I've got it written down here. Yeah, 15 years of self-hosting.

Richard Campbell: Wow.

Paul Bone: It's quite an old project. It started 17 years ago.

Carl Franklin: So let's get back to what kinds of problems can it solve best. I'm still not figuring this out.

Paul Bone: Yeah. We're using Mercury for general purpose programming. It's just that a lot of what a programmer is doing is logical. So we find that a programming language that shows them the logic of what they're writing helps a lot in general.

Carl Franklin: Are we talking about a line of business applications? Are we talking about scientific calculations?

Paul Bone: Scientific calculations. They've been a little curious for a type but suddenly business applications are. We know of people using Mercury as a business rules engine to build up rules about different conditions. Say the client is retired and has a hundred of this size paid for. This insurance package is suitable for them.

Richard Campbell: Oh, I see.

Carl Franklin: So it's good for sort of querying data, would you say?

Paul Bone: Yeah. It is good for that. But I guess the message I'm trying to say is that it's good for almost everything.

Carl Franklin: Yeah.

Richard Campbell: This is a general purpose language although I don't think most developers think about parallel execution in their day-to-day general purpose programming.

Paul Bone: Yes. So that's something I'd like to come to. Because of how Mercury is organized, we're able to automatically parallelized programs in Mercury.

Richard Campbell: I see. So the idea is that the programmer doesn't need to know it's executing in parallel. It's just going to happen.

Paul Bone: Yes. Yeah. On the program I can say please make it execute in parallel, but they don't have to know how.

Richard Campbell: Right.

Paul Bone: When you use an optimizing compiler, you tell the compiler to optimize your program harder. It's exactly the same principle.

Richard Campbell: Okay.

Carl Franklin: And where does functional programming fit in to this?

Paul Bone: So Mercury also supports functional programming in that some of the basic concepts that Mercury has predicates and functions.

Carl Franklin: Yeah.

Paul Bone: A function is a predicate that can only succeed in one way. It only has one end for any set of inputs.

Carl Franklin: So let's break this down a little bit for the nonfunctional, non-logical programmer or listeners out there. Let's go down to brass tacks here. Predicate, define that for me.

Paul Bone: Okay. So predicates. I mentioned earlier the honed clause where you say this thing is true if they spot it at all sector. That can also be almost thought of as a predicate. It's simple enough to say that it's a predicate. So a predicate is something that you can prove to be true for its parameters, the grandparent example I used earlier. So a grandparent would take two parameters, the child and the grandparent, and grandparent is true for valid pay as opposed to arguments but it doesn't talk about whether it's computing the child from knowing that their grandparent is Jim or whether it's computing that the grandparent is Jim when the child is born.

Carl Franklin: Wow. I must be stupid. I still don't know what a predicate is. I just want you to define...

Paul Bone: It's very difficult to explain it without pictures.

Carl Franklin: Right. Let me just try to regurgitate what I hear. So it's a set of conditions that can prove something or a set of givens that can prove something.

Paul Bone: Kind of. A predicate is a piece of card that's true. I'm sorry.

Carl Franklin: It's really hard talking about codes sometimes, isn't it?

Paul Bone: Yeah.

Carl Franklin: I'll go look up and see what somebody else says here.

Paul Bone: Yeah. So it's a mathematical, like that of a calculus.

Richard Campbell: Predicates and logic are statements that are true or false depending on what their values are.

Carl Franklin: Here's what Wikipedia says on predicate logic. Predicate logic is the generic term for symbolic formal systems like first-order logic, second-order logic. That doesn't help. This formal system is distinguished from other systems in that its formulae contain variables which can be quantified.

Paul Bone: That's all true.

Carl Franklin: Two common quantifiers are the existential - "there exists," and universal - "for all" quantifiers.

Paul Bone: Yeah.

Carl Franklin: Yeah. I still don't know what a predicate is.

Richard Campbell: So for all grandparents, there's a grandchild.

Carl Franklin: Okay.

Richard Campbell: And if grandchild = Bobby then grandparent = Jim.

Paul Bone: Yes.

Carl Franklin: Okay.

Paul Bone: But Bobby may have more than one grandparent.

Richard Campbell: Right.

Paul Bone: He may have Grandma Nancy and so the pair like grandparent of Bobby and Nancy would also be true. If you would have a called grandparent that's specified to Jim as a first and I even then specified some X for the second argument, it would return either Jim or Nancy, or Jim and then Nancy when the code will execute the second time for whatever reason.

Carl Franklin: Okay.

Paul Bone: Because things are true.

Richard Campbell: Well, and you get the hint parallelism there when you simply refer to the grandparents. So Jim, however many they are, is irrelevant and could easily execute in parallel.

Paul Bone: That's true. That's what's called all poll parallelism because you're looking at what's -- because when you build up how grandparent works, you might either follow the mother or father link in the family tree and those things are disjunctive. So it's known as all parallelism because of the disjunction there.

Carl Franklin: Yup.

Paul Bone: I've lost you again.

Carl Franklin: Oh, yeah. And I'm sure our listeners are going eek all their heads too, some of them. I don't know, maybe it's just me.

Paul Bone: I hope we could cut a fair bit of...

Carl Franklin: Oh, no. It's good fun.

Paul Bone: Yup.

Carl Franklin: All right. So I also see from Wikipedia, which is of course is the answered source of all truth, that it's sometimes called first-order logic. Is that also true predicate logic?

Paul Bone: That's also true.

Carl Franklin: Yeah.

Paul Bone: Although Mercury supports higher auto logic as well.

Carl Franklin: So it looks like it's a logical statement that contains variables the outcome of which is not known until those variables are set.

Paul Bone: Yes.

Carl Franklin: But the logic is true no matter what. Is that right?

Paul Bone: Yeah, that's right.

Carl Franklin: Oh, my God. Oh, my God. I learned something.

Paul Bone: You'd be relieved to know that most Mercury programs don't actually use all of these stages. Most of them are very simple and they're what's called deterministic.

Carl Franklin: Okay.

Paul Bone: So that means that unlike Bobby who has up to four grandparents, it means that for any given input there's exactly one answer.

Richard Campbell: All right.

Paul Bone: It's not easy in a family example. Let's use you use the example of my whole program. So the first predicate that the Mercury runtime system calls when it starts your program is the main predicate which runs your whole program just like in C, and mine is true for a valid execution of your program and there's only -- in the deterministic program there's only one valid execution.

Richard Campbell: Right. This is like the square root of nine will always return three.

Carl Franklin: I see.

Paul Bone: Right.

Richard Campbell: As opposed to a nondeterministic function like what is the time.

Paul Bone: That's right.

Richard Campbell: Which each time you're asking you get back a different value.

Carl Franklin: Yeah.

Paul Bone: Yeah. If you would give it the parameter I'm asking you now when you ask what is the time, then that value of now there's still only one answer which is how we get around the problem of getting a file on the disk or reading input from the user and so on because if I open the file on the disk it might exist but then if I close it again and open it later somebody might have deleted it in the meantime and that would be a different result.

Richard Campbell: Sure.

Paul Bone: For what looks like the same input. So, to avoid those problems and actually be able to program practically, we pass around what we call the I/O state which represents the world outside the program.

Carl Franklin: Yeah.

Richard Campbell: It's your nowness.

Paul Bone: Yup. So if you don't see the variable in the program you know that it can have no effect on the outside world which is a really lovely thing to debugging.

Carl Franklin: Whoa. What a really strange and beautiful way to program. I mean, it's sort of sinking in that it's a totally different way to think about it, about how to interact with the computer.

Richard Campbell: Yeah. It's only penetrating me to the point where I'm just getting chills about it.

Carl Franklin: Yeah. I'm with you, buddy, I'm with you. I think a lot of C#, VB.NET business developers that listen to the show sort of hopefully build that same twinge of I'm beginning to get it.

Paul Bone: Yeah. Can I run through another example?

Carl Franklin: Yeah.

Paul Bone: So my favorite example is a Random Number Generator. Everybody knows that you call -- I mean I don't know C# but I know C. In C you would call REN with another parameter and you keep getting back different numbers. So that to be able to work what we have to do is pass the current state of the Random Number Generator and when it returns it not only gives you your random number but it has to give you the new state of the Random Number Generator which you'd use next time you call them.

Carl Franklin: What I know about Random Number Generator is they're not random. They're based on some number, usually a number of ticks that have happened since a certain time which is a number that's big and changes all the time, scrambled up, moved around, mathematized, etc, and that's used as a seed to generate a pseudo random number. But random number, I know that it's a very academic thing and people say, oh, that's not random.

Yeah, because if the seat is the same you're going to get the same number through the Random Number Generator.

Paul Bone: And each time you call REN, the state or the seat changes.

Carl Franklin: Yeah.

Paul Bone: But of course if you want something to change in purely declarative programming, you have to be able to see a variable for it in the source code that you're writing otherwise you know that it can't possibly change so we have to see the state of the Random Number Generator being passed around.

Carl Franklin: I see.

Paul Bone: So this is really cool when you start working with data structures. If you're managing a dictionary and you insert an item into a dictionary, what you get back is a new dictionary. The old one can still exist and then you start working with the new dictionary. You might want to delete a different item out of the dictionary. The previous versions of the dictionary still exist in memory.

Richard Campbell: Right.

Carl Franklin: So they're immutable.

Paul Bone: Yeah. And now the memory between them is actually shared like the items themselves and often much of the structured dictionary. So this becomes really cool when you want to implement undo. All you have to do is revert it.

Carl Franklin: Oh, sure.

Richard Campbell: Because everything is still there.

Paul Bone: Yeah.

Richard Campbell: But this is also part and parcel with best practices with parallelism because as soon as memory is mutable you now have raised conditions and blocking and so forth to protect memory. If it's immutable, you're just writing new copies of things so there's no conflict between multiple threads executing on it.

Carl Franklin: And copying all that data, you know, memory is cheap but does it get expensive when those data structures are huge?

Paul Bone: Not when the data structures are huge, but when you make many modifications to the data very quickly that's when -- because often when you make a small modification to a lot of data, it only allocates a few cells.

Carl Franklin: Oh, sure.

Paul Bone: If you've got a binary tree and you delete or insert a new item, on average it only modifies or reallocates login items in the tree so it's just much smaller than the amount of memory that the tree uses anyway.

Carl Franklin: So that means even though it's immutable and you get a new collection, the items in that collection are still shared.

Paul Bone: Yes, that's right.

Carl Franklin: So it's really the metadata about the collection, the list, that's immutable.

Paul Bone: Yup. Well, that's not even mutable. You get a new version of the collection and you can still see the old collection.

Carl Franklin: Yeah.

Paul Bone: You still got a reference to it if you want to keep that old reference around. If you like all of the old reference, the garbage collector will get it.

Richard Campbell: I see.

Carl Franklin: Now also in Mercury is kind of a weird idea but I'm very curious to find out how it works. Declarative debugging.

Paul Bone: Yeah. So that works using the same ideas because nothing in memory is ever changed and you have to -- for something to have changed, you need to explicitly be able to see it in the program, so all state is explicit. Then any node in your call graph represents that part of the program like that's some call graph of a program. Are you still with me?

Carl Franklin: Okay.

Paul Bone: Yeah. I mean are you following?

Carl Franklin: So far.

Paul Bone: Yeah. So, if the debugger can ask the programmer, "Hey, see this node in the call graph? Does it look good to you?" and the programmer can say yes or no. If the programmer says no, that node in the call graph, there's a problem, then an automatic tool, what we call a

declarative debugger can search below that node in the call graph to its children, the things it calls and ask the same question of nodes and this can help the programmer assign where the bug is.

Richard Campbell: So essentially they're stepping backwards through all those iterations until they find when things made sense and the transition from when they made sense to when they didn't is where the bug lives.

Carl Franklin: So that's a very nice way to pinpoint problems.

Paul Bone: Yeah. What we'd like to do, I don't know if we've -- I know that somebody has worked on this in the past, but I don't how complete the support for this is. But this is going to test weight and let's say you've got a thousand tests which is pretty normal, you might have some that pass and some that fail and if you have coverage data for each of those tests and you can write no in the call graph whether the test executes and passes old files or pass more often than it fails and so on, you know more data are in the way your bug is and you don't necessarily need to ask programmers so often do you think the bug is in this part of the program?

Richard Campbell: Right.

Paul Bone: So I mean I'd like to say these programs debug themselves and they should be able to probably do 90% of the debugging themselves.

Richard Campbell: But then, yeah, your declarative part will be basically identifying the intended state at each of the iterations.

Paul Bone: Yeah. So that's something that an automatic program can never do. It depends on its weight. It has an idea about which things relate to working, which things relate to right result and which things relate to the wrong one.

Carl Franklin: Wild.

Richard Campbell: Yeah, I know. I'm getting this weird recursive thought like, well, if you know where my bugs why don't you just write the code in the first place.

Carl Franklin: That's right. What do you need it for? Yeah.

Paul Bone: Yeah. And so that's it, it's because the whole philosophy behind this is that you only need to tell the computer what the problem is that you're trying to solve and not have to solve it.

Richard Campbell: Right.

Carl Franklin: That is weird.

Paul Bone: That goes back to that idea.

Richard Campbell: Well, yeah. And so then the declarative debugging part of this is saying this is the solution I expect you to get to figure out where you went wrong because that is not the solution you gave me.

Paul Bone: That would be great, yeah. So I think we're not there yet, but...

Carl Franklin: What is automatic parallelism?

Paul Bone: So yes, it's the system part of my research but first I'd like to -- so we saw earlier that without side effects, without -- that it's easy. We saw that it's very easy to determine if running two things in parallel is safe because we can see what the dependencies they have on one another and whether state can change when they don't expect it and so on and therefore whether it's safe to run them in parallel or not. So determining whether something is safe is trivial in Mercury and the hard part is determining what things should be run in parallel to make the program more efficient.

Carl Franklin: So when you say it's trivial to determine if something is safe, does the safetiness of a variable, an object, whatever you call them, does that change as the program changes or is known from the beginning?

Paul Bone: It's known during compile time.

Carl Franklin: Okay.

Paul Bone: So it isn't something that the programmer can look at the source code and see very obviously.

Carl Franklin: So you have this parallel conjunction operator, the upper sand. So a conjunction, that's conjoining two things together?

Paul Bone: Yup. So in the grandparent examples that we were using, that has to find with a -- let's say to prove that X is his grandparent, to find some Y which is the first parent of X and then find a Z which is the first parent of Y. So those two call the parents conjoined. You have to satisfy both systems in order to satisfy grandparent.

Carl Franklin: Right.

Paul Bone: Which is what we mean by conjunction.

Carl Franklin: Okay.

Paul Bone: So a parallel conjunction is just one that says try and improve both of these things at the same time.

Richard Campbell: I'm not worried about what order they're executing.

Paul Bone: Yeah.

Richard Campbell: But that's declarative parallelism and that's clearly the developers saying you can do this in parallel.

Paul Bone: Yes. The benefit there is that the programmer doesn't need to worry about locking.

Richard Campbell: Yeah, no memory protection. No mutex. It's none of that stuff.

Carl Franklin: So automatic parallelism then avoids the problem of the programmer having to know where to optimize.

Paul Bone: That's exactly right. So when many programmers are asked to optimize their programs, the smart programmers reach for profilers because people know that programmers aren't very good at taking and naming the parts of their program that contribute the most to its execution like the slowest parts that are worth optimizing.

Richard Campbell: Right.

Paul Bone: So we use profilers to show us what things should I optimize.

Richard Campbell: Yes. Where is my program spending its time?

Paul Bone: Exactly. Usually by looking at 1% of the program and optimizing that, you can get 90% of the benefit or something like that.

Carl Franklin: Yeah.

Richard Campbell: Pareto's Law applies.

Paul Bone: Yeah. So what we want to do is use the same concepts for parallelizing programs because parallelism is essentially an optimization.

Richard Campbell: Right.

Paul Bone: It's difficult for the compiler to know which parts are going to be slower than others that's why we need to use the profiler here. So the programmer would compile that program for profiling, run it on some test data which would give them a profile and they can give that profile an automatic parallelization tool.

Carl Franklin: Wow.

Paul Bone: That tool will be able to look at the profile of the program and understand where the hotspots are and find places where there are two or more things that can be done in parallel and that are costly enough to be worthwhile doing in parallel.

Richard Campbell: Because there's an overhead to parallelism and if it doesn't really give you much benefit it will actually slow things down.

Carl Franklin: Yeah. There's such a thing as too much of a good thing.

Paul Bone: Yeah. If you parallelize too much of your program, I mean you've got what? A four-core, eight-core machine? If you parallelize too much of it you'll have 1,000 or more independent little tasks to do and only eight cores to do them on.

Richard Campbell: Right. And your processors are going to spend most of their time context switching to execute each of those tasks and they're actually doing the work.

Paul Bone: Exactly. So what people have done -- I mean parallel, automatic parallelism has been a research topic in the past, as you've said probably well back in the 1980s when people were looking at these men, but the mistake that a lot of people made was to parallelize too much of the program. We would have parallelized only a couple of places and only the places that give you the most benefit.

Carl Franklin: At Franklins.Net right now, you can get a DVD with over 11 hours of Billy Hollis on Silverlight 4.0 or 14 hours of Sahil Malik on SharePoint 2010 each for only 695. Order online at www.franklins.net. Are you looking to change jobs? Infusion Development has offices in New York City, Toronto, London, Dubai, and Poland. Infusion has hired a whole handful of happy .NET Rocks listeners. Contact me for an introduction at carl@franklins.net.

Richard Campbell: Well, the other thing is it in the 1980s multiple core machines were incredibly rare and expensive. You only had one core and it was a pretty simple one at that and parallelism just didn't do much for you.

Paul Bone: And only in the consumer's field there were some heavy ion service that had many cores...

Richard Campbell: Right.

Paul Bone: At lease they weren't coarse. They were old chips or old cards, I'm not sure. I'm speaking beyond my years there.

Richard Campbell: But I just think this has become so much more relevant because Intel is shipping experimental quantities of 80-core processors in one chip.

Paul Bone: Yeah. And I asked Intel for a 48-core chip and they said no.

Carl Franklin: Oh.

Richard Campbell: Ooh.

Paul Bone: But yeah, that would have been an awesome test bed.

Richard Campbell: You guys aren't cool enough for a 48-core chip? Because this is pretty cool.

Paul Bone: Well, we actually looked at the 48-core chip and found that it didn't have any hardware level case coherence mechanisms which means that the way Mercury is written at the moment, especially the garbage collector which is we've borrowed the bone garbage collector, which is a popular conservative garbage collector for C#. The bone garbage collector and the Mercury runtime are written to assume that the machine has some kind of cache coherence and not programming without a cache coherence is more than difficult. So, it's something that we need a couple of use to get ready for.

Carl Franklin: Case coherence, is that what you said?

Richard Campbell: Cache.

Paul Bone: Cache.

Carl Franklin: Oh, cache, right.

Paul Bone: Oh yeah, sorry. American people say cache.

Carl Franklin: Yeah.

Paul Bone: In Australia we say cache.

Carl Franklin: Okay. That's radical, dude.

Paul Bone: Right.

Richard Campbell: So one of the big challenges when you actually start to execute things in parallel is that the debugging gets so much more complex.

Paul Bone: Yeah. Well, the debugging, because parallelism is deterministic in Mercury, debugging isn't an issue. Not for getting the correct result out of your program, but to make sure that it executes efficiently is difficult.

Richard Campbell: There are interesting tools in the latest version of Visual Studio that help you see whether you're really executing effectively in parallel. The task you've broken up, do more than one thing and run it at the same time even if four things run at once, the three of them wait around while one of them finished. I mean that whole thing can be visualized in a profiler.

Paul Bone: Yeah, that's exactly right. So this is where I hope you are going. So we've found barge for the lack of a better word, a visual part file of it has been developed for Haskell, for parallel Haskell programs.

Carl Franklin: Nice.

Paul Bone: This was convenient because Haskell's runtime system is similar to us so we've been able to adopt it to work with Mercury.

Carl Franklin: That's very cool.

Richard Campbell: We never give a lot of love to Haskell.

Carl Franklin: Well, we did. We did a show on Haskell, did we not?

Richard Campbell: Yeah. The problem is that you just said it services only .NET space. I mean, Mercury at least has a C# library of some kind.

Carl Franklin: Ping, you just said the magic word. I was wondering what was the connection between .NET and Mercury.

Paul Bone: Yes. So if you want you can tell the Mercury compiler rather than to generate C code or Java code which it supports, you can tell it to generate C# code which then it will use Microsoft tools to compile.

Richard Campbell: Wow.

Paul Bone: I haven't done it myself.

Carl Franklin: How? How does...? How do you get from a functional declarative language to an imperative expression?

Paul Bone: The Mercury compiler just decides how best to execute your code. It's the same way we go about generating C code.

Carl Franklin: So is it going to use threads.

Paul Bone: Yeah.

Carl Franklin: It is.

Paul Bone: So it would use the -- I'm not sure in .NET. I use threading in Java. Is it similar to that in which you use a thread object?

Richard Campbell: Yeah.

Carl Franklin: Yeah. Well, there are a lot of great new parallel tools in .NET as well and I would hope that it would use some of that stuff like the Task Library.

Richard Campbell: Yeah. But I'm betting that's too new for this.

Paul Bone: It might be too new. So there's a thread library analysis standard, there's a thread module in analysis standard library which if that supports C#'s threading stuff then you're good to go. You unfortunately don't get to use parallel conjunctions but you do get to use more explicit parallelism. So the parallel conjunction operator and the automatic parallelization, these are only supported on the low level C backend unfortunately.

Carl Franklin: Okay.

Paul Bone: I'm sorry to disappoint you.

Carl Franklin: No, no. That's all right. I know this sound like I'm just asking the same question but give me some more practical uses that a line of business developer can take Mercury, generate C# code and generate code that's cleaner and better than something they could have written themselves. Are we talking mostly in middle tier code here?

Paul Bone: Yeah. That would be where I'd use it. I know of a business that uses Mercury in order to interface with both Java and .NET code. The clients may have libraries that are written in Java or in .NET and then they've written, like I've said before, their business rules engine where they choose which insurance or which plan like a mobile phone plan the customer is eligible or best suited for using NMF and that engine is written in Mercury. And then they want to integrate that with the software that the company already has which maybe in .NET or in Java. So this is mission critical.

Carl Franklin: Yeah.

Paul Bone: Yeah. I don't know if you've heard of them.

Carl Franklin: No, I haven't.

Paul Bone: The C# backend is actually their work. So they have contributed that to Mercury.

Richard Campbell: That's actually this company that's been using it, they wanted to have it work in IL and so they went to the trouble of actually building themselves.

Paul Bone: But they build the C# backend. Previous to that we did have a .NET IL backend.

Richard Campbell: Oh, I see.

Paul Bone: Which was a separate project. The Mercury project was given a grant by Microsoft to build that. Unfortunately, now the .NET's immediate language has moved on in versions and so we're no longer compatible.

Richard Campbell: Oh, I see.

Paul Bone: There's nobody currently maintaining that. So it's not useful at the moment because .NET has moved on.

Richard Campbell: So the old IL version that you had is broken, but the C# version works.

Paul Bone: That's right.

Richard Campbell: Okay.

Paul Bone: And the idea of the we chose to use C# in the second version rather than in your IL version because C# is less likely to change as Microsoft...

Richard Campbell: Yeah. You'd get caught up with the same problem. It will get broken again.

Paul Bone: Exactly.

Carl Franklin: Right.

Paul Bone: So I mean they want to keep C# the same because that way people can still use their programs. So it makes sense to be compatible with C# rather than IL.

Richard Campbell: I guess the question is why would I use this over the Task Parallel Library or anything in the new .NET 4.0 parallelism features?

Carl Franklin: And I think you probably are not going to really understand that until you get your hands on it and actually see the code.

Paul Bone: Yeah. I've passed certainly those tools. So I'm tentatively guessing those tools are probably great if you know the problem that you're trying to solve decomposes well for parallel execution. Say it's image manipulation or right tracing or something like that that decomposes well, you're probably best off using a library like that where Mercury sorts our parallels and _____ 47:37 it's not obvious to the programmer how they should parallelize something and that's the problem we're trying to solve. It's not productive to give programmers tools that are equivalent to those they already have.

Richard Campbell: Yeah.

Paul Bone: So, what we're building here is does data parallels.

Richard Campbell: So the goal is automated parallel always.

Paul Bone: Yeah. So it shouldn't matter what the shape of your computation looks like to the automatic parallel relation tool.

Richard Campbell: They shouldn't be in the operative world.

Paul Bone: Even if the program in the future and the way it would get parallelize changes, you just rerun the analysis tool and it's parallelized again without you having to do any of it to update it.

Richard Campbell: Now is this actually your PhD thesis? Is this how it's going to happen?

Paul Bone: That's right.

Richard Campbell: The question is, are you going to use your PhD?

Paul Bone: Well, whether I succeed or not I believe that I'll be successful in getting a PhD. I may be able to prove why doing this is impossible.

Richard Campbell: Right.

Paul Bone: And then save other researchers the time just looking at it. I don't think that's likely.

Richard Campbell: Right.

Paul Bone: But it's something that could happen.

Carl Franklin: You know the quest for truth and understanding is wonderful when you know that whether you succeed or fail, you succeed.

Paul Bone: Yeah. Because either way you learn something.

Carl Franklin: That's right.

Paul Bone: The risk that's just about finding out something that nobody on this earth knew before.

Carl Franklin: Yeah.

Paul Bone: That's the thing that makes my spine tingle.

Carl Franklin: Yeah, absolutely.

Richard Campbell: Contributing to the science.

Paul Bone: Yeah, that's it.

Carl Franklin: Tell us a little bit about the ICLP 2011. What is that event again?

Paul Bone: That's the International Conference of Logic Programming. It's held in Lexicon, Kentucky this year.

Carl Franklin: What kinds of things go on at a logic conference?

Paul Bone: This is very much an academic conference. People will bring their papers about different ideas in logic programming and related fields and present them there. So I'll be presenting my work that I've done on trying to calculate whether a particular parallelization is beneficial or not due to dependencies between the parallel tasks.

Carl Franklin: Do you ever get the feeling your brain is just going to explode?

Paul Bone: Actually no.

Carl Franklin: No. It tickles. It feels good.

Paul Bone: The more and more I try, the more I realize I don't know. Yeah, it's very much that I don't know how to do it yet.

Carl Franklin: It's so cool.

Richard Campbell: There's always more.

Paul Bone: Yeah. So I'm really excited about presenting there because it will be chance for

me to meet other researchers and find out what they're doing and also get their feedback on my work so it's a big deal for me.

Carl Franklin: So let's call out the website for Mercury. You made a TinyURL for us?

Paul Bone: So it's tinyurl.com/mercuryproject.

Carl Franklin: Awesome.

Paul Bone: And you can find documentation, and downloads, and research papers there.

Carl Franklin: This will run on MPC?

Paul Bone: Yeah. This runs on Windows and Mac and Linux.

Carl Franklin: All of the above, huh.

Paul Bone: And probably all the types unique to it. I don't think we've tried lately.

Carl Franklin: When your website talks about backend, that means this is the code that it generates. Right?

Paul Bone: The Mercury compiler can generate two types of C code, the low level and high level C, they're just different strategies for getting to the same place. It can generate Java and C# code as well.

Carl Franklin: And Erlang?

Paul Bone: I think it generates Erlang. I can't remember how polished that backend is.

Carl Franklin: Yeah. It says it's in beta.

Paul Bone: That's the matter with my memory mold on the backend itself.

Carl Franklin: The website says it's in beta.

Paul Bone: Okay.

Carl Franklin: One guy out there is going, oh, damn.

Richard Campbell: My Erlang.

Paul Bone: Yeah.

Carl Franklin: So cool. And then native code, that compiles to assembler?

Paul Bone: We do that through the low level same backend or through any of the same backend. We generate -- the low level code generator gives you that look like assembly code. It's an abstract machine code for the Mercury abstract machine which is then compiled using the end. A lot of nasty use the same preprocessor and so then you can generate the same code from there. The download is on the website. When you download the source code from Mercury, it already contained precompiled free code in there so you don't need a Mercury compiler installed to install Mercury compiler.

Carl Franklin: Okay. It seems to me that if you're doing that low level C or assembler or the inline assembler that the C compiler, pre-compiler creates, you're generating one heck of a performant power house of a program.

Paul Bone: Yeah. Mercury is pretty efficient. We made all the other prologues, not that Mercury is a prologue. We made prologues for performance and we haven't compared it against other languages lately. I heard anecdotal reports that Mercury cards can get close to the performance of C and I've also heard anecdotal reports that it will be Java.

Carl Franklin: Well, we should mention one more thing which is that your research is funded.

Paul Bone: That's right.

Carl Franklin: Let's give some props out to those who make this possible.

Paul Bone: So yeah. I like to thank the Australian government for my Australian post-graduate award scholarship and National ICT Australia, my top op scholarship. So they have to be thanking for me being able to spend a significant part of my life working on this.

Carl Franklin: That's fantastic. Thanks to them very much.

Paul Bone: Yeah.

Carl Franklin: Well, do you think you might want to do a dnrTV show on this to show people exactly what this looks like?

Paul Bone: Yeah.

Carl Franklin: If my brain doesn't explode.

Paul Bone: It will definitely be easier with pictures.

Carl Franklin: Okay.

Paul Bone: It may even be great with a demo.

Carl Franklin: Awesome. Let's make that happen.

Paul Bone: Yeah. If August is okay with you, guys, that would be -- or like July or something.

Carl Franklin: Absolutely. Paul, is there anything else that you want to say before we call it a show?

Paul Bone: Oh, that's right. I want to thank you for inviting me to the show.

Carl Franklin: Oh.

Paul Bone: Yeah.

Carl Franklin: It's out pleasure.

Paul Bone: Yeah. Thank you to your invitation to speak on .NET Rocks! It's been a pleasure speaking with you and having my brain tick about Mercury.

Carl Franklin: Well, I'm sure our listeners really appreciate it. Check the website, the .NET Rocks! website because people do leave comments and they may have questions for you.

Paul Bone: Cool.

Carl Franklin: All right, great.

Paul Bone: All right.

Carl Franklin: Thank you, Paul. Thank you for listening, dear listeners. We'll see you next time on .NET Rocks!

[Music]

Carl Franklin: .NET Rocks! is recorded and produced by PWOP Productions, providing professional audio, audio mastering, video, post production, and podcasting services, online at www.pwop.com. .NET Rocks! is a production of Franklins.Net, training developers to work smarter and offering custom onsite classes in Microsoft development technology with expert developers, online at www.franklins.net. For more .NET Rocks! episodes and to subscribe to the podcast feeds, go to our website at www.dotnetrocks.com.