Nom et prénom du coordinateur / coordinator's name	**ZAPPA NARDELLI Francesco**		
Acronyme / Acronym	**WMC**		
Titre de la proposition de projet	Concurrence, mémoires faiblement cohérentes, et compilation certifiée		
Proposal title	Weak Memory Concurrency and Verified Compilation		
Comité d'évaluation / Evaluation committee	SIMI 2		
Type de recherche / Type of research	X Recherche Fondamentale / Basic Research ☐ Recherche Industrielle / Industrial Research ☐ Développement Expérimental / Experimental Development		
Aide totale demandée / Grant requested	204880 €	**Durée de la proposition de projet / Proposal duration**	48 mois

1. RESUME DE LA PROPOSITION DE PROJET / PROPOSAL ABSTRACT

Multiprocessors and multicore processors are now ubiquitous, but programming these systems, to deliver high-performance and reliable systems, is very challenging. Shared-memory is the programming abstraction exported by the hardware, and most multi-threaded programs communicate through memory shared between the threads. Traditionally concurrent execution was viewed as simply an interleaving of the steps from the threads participating in the computation. Thus if we started in an initial state in which all variables are zero, and one thread executes:

```
x = 1; r1 = y;
```

while another executes

```
y = 1; r2 = x;
```

either the assignment to `x` or the assignment to `y` must be executed first, and either `r1` or `r2` must have a value of one when the execution completes. However, it has proven impractical to guarantee such a restrictive memory behaviour, and mainstream programming languages (such as C, C++, Java) exploiting multithreaded hardware allow both `r1` and `r2` to remain zero in the above example. There are two reasons for this:

- for efficiency reasons, compilers may reorder memory operations if that does not violate intra-thread dependencies;

- the hardware may reorder memory operations based on similar constraints.

The forthcoming revision of the C++ standard (the C standard will be updated accordingly to preserve compatibility) specifies all the constraints that the possible outcomes of a parallel program must respect. This requires special architecture-dependent support in C and C++ compilers.

The goal of this grant proposal is to investigate the formal verification of realistic compilers for concurrent dialects of the mainstream languages C and C++. We will target both the x86 and Power/ARM architectures, which require radically different compilation strategies and proof methods, focussing initially on sample compilation schemes and then lifting these results to fully-fledged compilers. In addition we will design and prove correct novel compile-time optimisations for these languages and compilers.

2. CONTEXTE, POSITIONNEMENT ET OBJECTIFS DE LA PROPOSITION / CONTEXT, POSITIONNING AND OBJECTIVES OF THE PROPOSAL.

2.1. CONTEXTE DE LA PROPOSITION DE PROJET / CONTEXT OF THE PROPOSAL

Computer science is undergoing a difficult transition. Over the last 40 years we have seen exponential improvement in the performance of sequential computation. Now, however, the industry is hitting constraints, from power dissipation, CPU-memory bandwidth, and the

limits of instruction-level parallelism. Future performance increases must therefore come from increased concurrency, which is finally becoming mainstream. Multicore processors are now ubiquitous, with the number of cores in a typical laptop, server, or even mobile phone increasing rapidly.

However, programming these multiprocessors, to deliver high-performance and reliable systems, is very challenging. It has proved impractical to give large numbers of processors simultaneous fast access to a large *sequentially consistent* shared memory, in which (Lamport, 1979) "the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program". Instead, memory accesses of real multiprocessors may be reordered in various constrained ways, with different processors observing the actions of others in inconsistent orders, so one cannot reason about the behaviour of such programs in terms of an intuitive notion of global time. For a simple example, on x86 processors, given two shared memory locations `x` and `y` (initially holding `0`), if two processors `P0` and `P1` respectively write `1` to `x` and `y` and then read from `y` and `x`, it is possible for both to *read `0` in the same execution*.

P0		P1	
write	x ← 1	write	y ← 1
read	r0 ← y (0)	Read	r1 ← x (0)

This fundamental problem is exacerbated by four further issues. First, the concurrent algorithms that are now being developed, and which are key to exploiting multiprocessors (via high-performance operating systems, hypervisor kernels, and concurrency libraries), are very subtle, so informal reasoning cannot give high confidence in their correctness. Second, while there has been extensive prior work on software verification for concurrency (including model-checking, temporal logics, rely-guarantee reasoning, separation logic, and process calculi), almost all of it neglects such relaxed memory behaviour, instead assuming sequential consistency. Third, the vendor specifications of processor architectures, e.g. by Intel and AMD (x86), IBM (Power), and ARM, are informal prose documents, with many ambiguities and omissions; such informal prose is the industrial state of the art. Finally, and most importantly, this relaxed memory behaviour is a concern not just for low-level programmers, but is also partially exposed in high-level languages, e.g. with the problematic Java Memory Model (Manson et al., 2005) or the forthcoming C++0x standard (JTC1/WG14). For a simple example, it is easy to see that in a sequentially consistent execution the program below, where `r1`, `r2` are local variables, while `x`, `y` are shared, cannot print `1`.

P0	P1
r1 = x y = r1	r2 = y x = (r2 == 1) ? y : 1 print r2

However a modern compiler (in this case Sun Hotspot) might perform the following sequence of optimisations

P0	P1
r1 = x y = r1	r2 = y x = (r2 == 1) ? y : 1 print r2

⇒

P0	P1
r1 = x y = r1	r2 = y x = 1 print r2

⇒

P0	P1
r1 = x y = r1	x = 1 r2 = y print r2

and the resulting program can then print 1: *unexpected behaviours can be introduced by compilation* as well.

At the same time, this is an exciting time for the formal verification of software, in part because several threads of research, in progress for decades, have the potential to cohere. These include a gradual revolution in the specification methods for operational semantics of programming languages (1994-2010), the maturation of mechanized proof assistants (1978-2010), successes in compiler verification (1989, 2006), and progress in the specification of weak-memory models (2006-2010).

In this project I will build on these advances to address several issues related to the verified compilation of concurrent high-level languages.

2.2. ÉTAT DE L'ART ET POSITION DE LA PROPOSITION DE PROJET / STATE OF THE ART AND POSITIONING OF THE PROPOSAL

Research on weak-memory models traditionally abstracted almost entirely from the actual processors (Adve et al., 1996; Higham et al., 1997). In the last four years, in collaboration with Peter Sewell and others, I worked toward giving mathematically precise and experimentally validated *formalisations of the x86 and Power/ARM memory models and instruction semantics*. For x86 we produced both an axiomatic and operational model (formalised in HOL and Coq) (Sarkar et al., 2009; Owens et al., 2009; Sewell et al., 2010): contrary to the vendor documentation, it turns out that x86 behaves basically as Sparc TSO (a simple, well understood, model) and our formalisation can be considered as a reference by assembly programmers and compiler writers (Boehm, 2010). For Power/ARM the model is more complex: past attempts (Adir et al., 2003; Alglave et al., 2009; Alglave et al., 2010) do not respect faithfully the semantics of the barriers or accommodate load-reserve and store-conditional instructions. The effort is now converging toward an operational model that abstracts the internal micro-architecture of the processor, however work remains to be done to finalize the instruction semantics. The development of these x86 and Power/ARM models, which constitute the foundation of this proposal, has been partially supported by ANR-06-SETIN-010.

Meanwhile, in a remarkable tour de force, Leroy has demonstrated a *proved correct optimizing compiler* (called CompCert) from a dialect of C (called Clight) to machine language for Power and x86 architectures (Leroy, 2009). As part of this demonstration, Leroy specified an operational semantics for Clight, he specified an operational semantics for the Power (and x86) machine language, and he built a machine-checked proof in the Coq proof assistant that the compiler preserves behaviour from one operational semantics to another. Verified (or verifying) compilers is a long-term goal which we share with researchers worldwide, e.g. see the works of Benton (Benton et al., 2007) and Chlipala (Chlipala, 2007).

Generalising Leroy's result to a concurrent language, and dealing with the relaxed-memory-model properties we have discussed here, remains a major challenge. Hobor, Appel (Princeton) and myself begun to address the sequentially consistent case under the hypothesis that all shared-memory accesses in the source program are well-synchronised (e.g. protected by locks, a property usually referred to as *data-race freedom*) (Hobor et al., 2008). However most modern concurrent low-level high-performance code (ranging from simple spinlock implementations to fancy concurrent data-structures) relies on racy interactions, and this limits the scope of this approach. In collaboration with Sewell, Vafeiadis and others, I recently designed a concurrent C-like language, called ClightTSO, that exposes the processor model for high-performance code, and studied verified compilation from ClightTSO to x86, which we validated with correctness proofs (building on CompCert) for

the most interesting compiler phases (Sevcik et al., 2011). Although this research showed that it is possible to reason about the compilation process of racy programs, ClightTSO and its compiler CompCertTSO are not intended to define and implement a general purpose language. Moreover they are tied to a particular architecture.

For general purpose high-level languages, designing the memory model is a hard task in itself. Several languages, including OCaml, do not implement true-concurrency multi-threading at all and cannot exploit the parallel computation capabilities of modern processors. Others, including most scripting languages and mainstream languages as C and C++, rely on external thread libraries: there is widespread consensus that this is not the right approach (Boehm, 2005). Java had integrated multithreading since its first version but, by the year 2000, the initial specification was shown to allow unexpected behaviours, prohibit common compiler optimisations, and was challenging to implement on a weakly-consistent multiprocessor (Pugh, 2000). It was superseded around 2004 by the JSR-133 memory model (Gosling et al., 2005), but the resulting model (which attempts to solve a difficult challenge as it must ensure some memory safety requirements for all programs while enabling common optimisations) is quite intricate and, unfortunately, poorly understood. For instance it turned out that, standard optimisations as common subexpression elimination were illegal in the model (Sevcik et al., 2008). Such optimisations were claimed to be permitted, and are implemented by typical compilers including Sun's reference HotSpot compiler, as shown above. For C++, an ongoing effort, currently near completion, attempts to explicitly provide semantics for threads in the next revision of the standard (called C++0x) (Boehm et al., 2008; JTC1/SC22/WG14). The C standard (called C1x) will be updated accordingly to preserve interoperability. Partially following the Java experience, the revised standard gives semantics only to well-synchronised (data-race free) programs but does not attempt to provide safety guarantees about racy programs. However it includes an extensive framework, called *low-level atomics*, for low-level synchronisation with intricate semantics. The semantics of low-level atomics has been recently formalised by Batty et al. (Batty et al., 2011). This work also includes a proof of a sample compilation scheme for C++0x low-level atomics to the x86 processor.

2.3. OBJECTIFS ET CARACTÈRE AMBITIEUX ET/OU NOVATEUR DE LA PROPOSITION DE PROJET / OBJECTIVES, ORIGINALITY AND/ OR NOVELTY OF THE PROPOSAL

This projects aims at studying *novel compiler verification techniques* with the long term goal of building *verified optimising compilers for C++0x-like concurrent languages to x86 and Power/ARM architectures*.

In particular, building on our past work on the formalisation of the x86 and Power/ARM architectures, on the CompCertTSO verified compiler, and on the C++0x formalisation, we will:

1. work towards a certified compiler for a concurrent C-like language that integrates the C++0x memory model; and

2. design and study the soundness of compiler optimisations in the context of concurrent programming languages.

These results are needed: it has become widely accepted that industrial software developers need robust ways of programming and reasoning about multicore systems, and the subtle complexities of memory models make them ideal targets for mathematically rigorous methods.

3. PROGRAMME SCIENTIFIQUE ET TECHNIQUE, ORGANISATION DE LA PROPOSITION DE PROJET / SCIENTIFIC AND TECHNICAL PROGRAMME, PROPOSAL ORGANISATION

3.1. PROGRAMME SCIENTIFIQUE ET STRUCTURATION DE LA PROPOSITION DE PROJET/ SCIENTIFIC PROGRAMME, PROPOSAL STRUCTURE

The C++0x standard mandates that for a well-defined and interesting subset of the language, programs that do not contain data-races must have sequentially consistent semantic. Prototype implementations of the sequentially consistent atomic primitives have been proposed; the following table presents an x86 sample implementation by Terekhov (Terekov, 2008):

```
Load Seq_Cst:  LOCK XADD(0) // alternative: MFENCE,MOV (from memory)
Store Seq Cst:  LOCK XCHG    // alternative: MOV (into memory),MFENCE
```

and a Power prototype by McKenney and Silvera (McKenney & Silvera, 2010):

```
Load Seq_Cst:  hwsync; ld; cmp; bc; isync
Store Seq_Cst: hwsync; st
```

These compilation schemes are quite expensive (fences and locked instructions can consume 100s of cycles) and provide more synchronisation than needed for many concurrent idioms. On the one hand it is expected that a real-world implementation does compile time optimisations to get reasonable performance. On the other hand, the C++0x standard includes low-level memory-access primitives to be used in high-performance code by expert programmers: memory access instructions are parametrised by a memory order MO that specifies how much synchronisation and ordering is required. The strongest ordering is required for MO_SEQ_CST actions (which is the default, as used above), and the weakest for MO_RELAXED actions. In between there are MO_RELEASE/MO_ACQUIRE and MO_RELEASE/ MO_CONSUME pairs, and MO_ACQ_REL with both acquire and release semantics. For instance, consider the common programming idiom where one thread writes some data x (perhaps spanning multiple words) and then sets a flag y while the other spins until the flag is set and then reads the data:

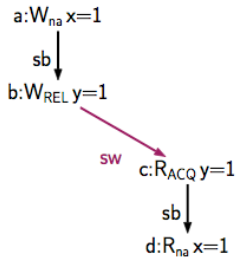
`// sender`
`x = …`
`y = 1;`
	`// receiver`
	`while (y == 0);`
	`r = x;`

The desired guarantee of this superficially racy program is that the receiver must see the data writes of the sender. This can be achieved with an atomic store of y annotated MO_RELEASE, and an atomic load of y annotated MO_ACQUIRE. Weaker memory orderings can be used for other common racy idioms, for instance read-consume pairs enable efficient implementation of algorithms that use pointer reassignment for commits of their data, e.g. read-copy-update (McKenney & Walpole, 2007). Again, sample compilation schemes have been proposed, for instance for Power we have:

```
Load Acquire:  ld; cmp; bc; isync
Store Release: lwsync; st
```

These sample compilation schemes are simple mappings from individual source-level atomic operations to small fragments of assembly code, abstracting from the vast complexities of compilation of a full C-like language. However reasoning about their behaviour involves reasoning about all the complexity of the C++0x memory model. Verifying that these prototypes are indeed correct implementations is crucial both for the design of the standard and to inform the design and verification of a compiler with memory-model-aware optimisations.

The statement of correctness of a compiler relates the behaviours of the compiled program to the behaviours of the source program. Contrarily to sequentially consistent execution for which the semantics can be expressed in terms of changes to a monolithic memory, an execution consists here of a set of *memory actions* and various relations over them, and the memory model axiomatises constraints on those. In the example above, we have that any instance of a read-acquire that reads from a write-release gives rise to a *synchronizes-with* (sw) edge that, together with the *sequenced-before* (sb) edges that capture the intra-thread evaluation order, results in the following (axiomatised) behaviour:



A complete description of the dependencies required to model faithfully the C++0x standard has recently been proposed (Batty, et al. 2011).

The CompCert compiler by Leroy et al. demonstrates that proving the correctness of a realistic compiler is feasible for sequential languages, and our work on CompCertTSO shows that Leroy's result can be extended to a concurrent language. However both CompCert and CompCertTSO rely crucially on operational reasoning, and it is unclear how much of their structure can be reused when the semantics of the source program is expressed as axiomatic relations. Novel techniques will be required to reason about compiler correctness on top of axiomatic models, and to propagate the various causality relations across the compilation phases and intermediate languages of a compiler.

Research goals: in Tasks 2 and 3 we will establish groundwork necessary for a full compiler verification for a concurrent C-like language that integrates the memory model of C++0x, both for Power/ARM and x86 architectures. The x86 sample implementation has already been proven correct by co-investigators of this proposal, but the x86 memory model makes this result much simpler than the equivalent for Power/ARM. Accordingly, in Task 2 we set out to prove the correctness of the Power/ARM sample compilation schemes. In Task 3 we tackle the ambitious goal of defining a fully-fledged C-like language with the C++0x concurrency model, and implementing a verified compiler for it. For this, we will target the x86 architecture, because its memory model is better understood and to avoid dependencies on Task 2.

There is a great potential to optimise the code generated by these compilation schemes. For an example of a simple optimisation on x86, a sequence of two `SEQ_CST` memory writes below is likely to be compiled as the code on the right:

```
x.write(1);              mov %eax, $1
y.write(1)               mov _x, %eax
                         mfence
                         mov _y, %eax
                         mfence
```

It is easy to convince somebody familiar with the x86 memory model that the first `mfence` instruction is redundant as the store buffer will be anyway flushed by the second fence, and no other instruction reordering can occur. A compiler might (and should) automatically detect such situations and optimise the redundant fences away. This, and similar, optimisations introduce unobservable nondeterminism and their correctness cannot be proved by a simulation argument, contrarily to the CompCertTSO phases.

Research goals: in Task 4 we will study compiler optimisations in the context of concurrent languages, and we will work out new techniques to formally verify their correctness.

From the user's perspective C++0x low-level atomics, or visibility of TSO reorderings, play a key role in programming and reasoning about high-performance concurrent algorithms. Although not directly focused on algorithm verification, this research project will be driven by the problems raised by the implementation and the verification of concurrent algorithms. Tool support is required to manage large semantics definitions: we will make our tools and infrastructure publically available (Task 5).

3.2. DESCRIPTION DES TRAVAUX PAR TÂCHE / DESCRIPTION BY TASK

The tasks below are organised thematically, not chronologically, and will run concurrently during most of the project. Detailed timing and dependencies are in the task schedule below.

3.2.1 TÂCHE 1 / TASK 1

Coordination

My collaboration with Peter Sewell's group at the Computer Laboratory of the University of Cambridge and with Viktor Vafeiadis (previously at Cambridge, now at MPI-SWS) is well established. We rely on shared repositories, daily teleconferences and frequent working visits, all of which are essential for effective collaborative work. Post-doc recruitment will be broadly advertised on international mailing lists. A web-page will report on the state of the project and will make our models, papers, and tools publically available. We will also engage with the community, with contacts with the C++ standard committee and GCC developers.

3.2.2 TÂCHE 2 / TASK 2

Sample Compilation of C++0X low-level atomics to Power and ARM processors

We set out to prove the correctness of the sample compilation scheme for C++0x low-level atomics proposed by McKenney and Silvera (McKenney & Silvera, 2010) for the Power architecture. A similar task has been tackled by Batty et al. (Batty et al., 2011) for the x86 architecture, but targetting Power raises new challenges as we shall see below.

1. Axiomatic Model of Power and ARM memory model

Memory models can be formalised in two styles: either operationally, by means of abstract machines, or axiomatically, defining valid executions in terms of memory orders. Each style has its own benefits: the abstract machine conveys the programmer-level operational intuition, while the axiomatic model supports constraint-based reasoning. For x86 we have both, with a proof of equivalence. For Power only an operational model that abstracts the processor micro-architecture has been defined and validated. We will produce an *equivalent axiomatic model* for the Power and ARM. Apart from being a valuable contribution on its own, an axiomatic presentation of the Power and ARM memory models is required to reason about compiler correctness (Task 2.2 and Task 3).

We will formalise a fragment of the instruction set large enough to be targeted by the backend of a realistic compiler. This must include the problematic instructions lwarx, stwcx and lwsync, which did not fit in previous formalisation of Power multiprocessors (Adir et al., 2003; Alglave, 2010). To increase confidence in the instruction semantics, we will build a robust and efficient infrastructure for testing the instruction semantics (Task 5).

2. Verification of the sample compilation scheme

To discuss the correctness of the proposed mapping in isolation, without embarking on a verification of some particular full compiler, we will work solely in terms of candidate executions and memory models. We will define an abstract compiler that maps the allowed candidate executions (in the sense of (Batty et al., 2011)) of a given program to Power executions (defined in Task 2.1) respecting the instruction compilation scheme but with some freedom in the resulting Power program order. We will then lift such Power executions to C++0x consistent executions: if this lifting exists we can conclude that the instruction mapping is correct.

We will consider increasing subsets of the C++0x low-level atomic standard, starting from MO_SEQ_CST and then covering all the weaker primitives: MO_RELEASE/MO_ACQUIRE pairs MO_RELEASE/MO_CONSUME pairs, and MO_ACQ_REL. The challenge here is that both the C++0x semantics and the Power/ARM semantics require respecting very particular dependencies, something that previous work on compiler proofs did not track. In addition the weaker semantics require complex instruction sequences (involving loops) to implement the synchronisation operations, e.g.:

```
Cmpxchg Relaxed, Relaxed:  _loop: lwarx; cmp; bc _exit; stwcx.; bc _loop; _exit
```

and will require novel techniques to reason about the potentially infinite executions resulting from unfolding loops in the context of memory models. The MO_CONSUME ordering has been included in the C++0x standard explicitly to exploit the particularities of the Power memory model: proving its correctness will establish a strong correspondence between the

formalisation of the C++0x memory model and the axiomatic presentation of the Power architecture (increasing confidence in both as a side effect).

3.2.3 TÂCHE 3 / TASK 3

Towards CompCert0x

Generalising the results of the previous section to a fully-fledged compiler remains a major challenge; the sub-tasks below are necessary intermediate steps. We will target here the x86 architecture, whose model is well understood and for which there exists a simple axiomatic presentation. Our approach is tailored so that we can reuse as much as possible of the existing CompCertTSO infrastructure.

1. Clight0x, and a compiler from Clight0x to x86

We will extend the Clight language by Leroy (roughly speaking a dialect of C without side-effects in expressions) with C++0x atomic types and instructions — we call the new language *Clight0x*. Even if we limit ourselves to the fragment without low-level atomics, defining its formal semantics is a challenge in itself as the complexities of the memory model (including the formalisation of data-race freedom) must be intertwined with the subtle aspects of a realistic C-like programming language.

Building on CompCert and CompCertTSO, we will implement a compiler from Clight0x to x86 assembler. Our first approach will be to add a translation phase from Clight0x to ClightTSO on top of CompCertTSO; this compilation phase will introduce memory barriers around all memory accesses, thus implementing MO_SEQ_CST, and will then rely on later phases to remove redundant barriers (Task 4). The proof of correctness will still have to deal with the mismatch between Clight0x axiomatic presentation and the ClightTSO.

In due course we will investigate fancier compilation schemes; however a proof of correctness of a sophisticated scheme is likely to have to propagate the axiomatic Clight0x semantics across several compilation phases. This is a challenging sub-task (Task 3.2) on its own.

2. Techniques to propagate dependencies across the intermediate languages of a compiler

The C++0x memory model is defined by an axiomatic model in terms of dependencies between events, and a semantic preservation proof cannot simply be done using simulations between the operational semantics of all the compiler intermediate languages, as was done in CompCert and CompCertTSO. We will study techniques to propagate and relate dependencies across the intermediate languages of our compiler of Task 3.1. This proof development may well drive re-engineering of the compiler. To make this task tractable, we will initially focus only on some selected, challenging, compilation phases, like the memory layout of the stack-frame or the allocation of shared data.

3.2.4 TÂCHE 4 / TASK 4

Correctness of compile-time optimisations for concurrent languages

We will investigate sound compile-time optimisations for concurrent languages such as ClightTSO and Clight0x.

1. Fence optimisations in CompCertTSO

Naive fence usage gives poor code quality and performance. Our goal here is to design, implement, evaluate, and prove correct, compiler optimisations that remove redundant memory barriers. CompCertTSO provides an ideal infrastructure to purse this research line, as the bare x86 memory model is lifted to all the intermediate languages of the compiler. These results can be used to optimise the naive compilation of the MO_SEQ_CST ordering of Clight0x. Initially we will focus on thread-local optimisations (that is, optimisations that are sound even if no assumption about the other threads are made). C++0x low-level atomics provide opportunities to enable more aggressive non-thread local optimisation, which we will study in relation to the advances of Task 3.

Although we do not yet have an infrastructure similar to CompCertTSO for the Power/Arm architectures, we will study fence optimisations for these weaker architectures as well. In particular, Power and ARM fences have a complex behaviour and their optimal placement is delicate.

2. Sequential optimisations and concurrent programming languages

A large body of data-flow analyses exists for analyzing and optimizing sequential code. Unfortunately, much of it cannot be directly applied on parallel code, because asynchronous updates break their correctness proof for the sequential case. We will investigate which sequential optimisation can be reused in a concurrent setting, possibly exploiting data-race freedom guarantees. We will focus both on the TSO memory model and the C++0x low-level atomic memory models, and implement the sound optimisations on top of CompCertTSO or the compiler for Clight0x of Task 3.

3.2.5 TÂCHE 5 / TASK 5

Tool Support for Large-Scale Semantics

This final task is devoted to infrastructure: engineering tools to work with the large mathematical definitions, of processor and language semantics, that are basis of the project. We will use existing proof assistants Coq, HOL, and Isabelle, and our Ott tool (Sewell, Zappa Nardelli, et al., 2010). In addition we need tools for translating processor semantics between those proof assistants (in general this is not feasible, but processor semantics tend to be type-theoretically simple, so it should be largely a matter of interfacing with the existing software). We would also like to take advantage of such translation to refactor Ott, and extend it to support richer forms of binding structure, functions, and semantic animation, to use for the high-level languages of the project.

Our processor semantics will have the semantics of instructions factored out from the memory model, but more work is necessary to make those semantics transparent to practicing software engineers, to the point where the informal prose describing instructions in processor manuals could be replaced by precise definitions.

We will also need to build more robust and efficient infrastructure for testing the instruction semantics and for exploring concurrent memory behaviour; these should be made into reusable tools.

3.3. CALENDRIER DES TÂCHES, LIVRABLES ET JALONS / TASKS SCHEDULE, DELIVERABLES AND MILESTONES

A timetable for the major components of my proposed research is given below.

Tasks	Year 1	Year 2	Year 3	Year 4
1: Coordination	XXXXXXX	XXXXXXX	XXXXXXX	XXXXXXX
2: Sample compilation of C++0x to Power/ARM				
2.1: Axiomatic model of Power/ARM	XXXXXXX	XXXXXXX		
2.2: Verification of the sample compilation scheme		XXXX	XXXXXXX	XXXXXXX
3: Towards CompCert0x				
3.1: Clight0x, and a compiler from Clight0x to x86	XXXXXXX	XXXXXXX	XXXXXXX	
3.2: Techniques to propagate dependencies		XXXXXXX	XXXXXXX	XXXXXXX
4: Correctness of compile-time optimisations				
4.1: Fence optimisations in CompCertTSO	XXXXXXX	XXXXXXX		
4.2: Sequential optimisations and concurrent languages		XXXXXXX	XXXXXXX	XXXXXXX
5: Tool support for large-scale semantics	XXXXXXX	XXXXXXX	XXXXXXX	XXXXXXX

Tasks 1 and 5 are support tasks that will run for the whole duration of the project. Each of the main tasks (Task 2, Task 3, Task 4) is ambitious and will span over several man-years of work. However it is possible to work on these in parallel, provided that enough human resources are available: for this reason I ask for funding of three man-years salary for PostDocs.

Deliberables and milestones			
Task		*Delivery date*	*Participants in charge*
Task 2.1	*Axiomatic presentation of the Power/ARM memory model*	T24. Progress report at T6, T18.	FZN, LM, PS
Task 2.2	*Verification of the Power/ARM sample compilation scheme*	T48. Progress report at T24, T36.	FZN, LM, PS
Task 3.1	*Clight0x and a compiler from Clight0x to x86*	T36. Progress report at T12, T24.	FZN, PS
Task 3.2	*Techniques to propagate dependencies across the intermediate languages*	T48. Progress report at T24, T36.	FZN, PS
Task 4.1	*Fence optimisations in CompCertTSO*	T24. Progress report at T12, T18.	FZN, VV
Task 4.2	*Sequential optimisations and concurrent programming languages*	T48. Progress report at T24, T36.	FZN, VV
Task 5	*Tool support for large-scale semantics*	T48. Progress report at T12, T24, T36.	FZN, LM, PS

(FZN: Francesco Zappa Nardelli; LM: Luc Maranget; PS: Peter Sewell; VV: Viktor Vafeiadis).

4. STRATEGIE DE VALORISATION, DE PROTECTION ET D'EXPLOITATION DES RESULTATS / DISSEMINATION AND EXPLOITATION OF RESULTS, INTELLECTUAL PROPERTY

We will publish our results in the peer-reviewed scientific literature, initially in conferences such as CAV, CONCUR, DAMP, ESOP, ICFP, LICS, PLDI, PODC, POPL, PPoPP, SOSP, and ITP. The broad scope of the project, covering several different subfields, will demand and enable particularly broad dissemination. We will make our semantic models, tools, and implementations publically available, via the web, as early in the project as possible.

This programme provides an unusual opportunity for fundamental research to have a broad and direct impact, in industry and academia. The results are needed: it has become widely accepted that industrial software developers need better ways of programming and reasoning about multicore systems, and the importance and subtle complexities of memory models make them ideal targets for mathematically rigorous methods. There is keen interest from processor vendors (witness discussions with architects at ARM, Intel, AMD, and IBM), and from OS, algorithm, and library developers (witness discussions with Lea, Harris, and McKenney); this project should enable a wide range of research on programming language design, compilation, verification, and algorithms, taking relaxed memory into account.

5. DESCRIPTION DU PARTENARIAT / CONSORTIUM DESCRIPTION

5.1. DESCRIPTION, ADÉQUATION ET COMPLÉMENTARITÉ DES PARTICIPANTS / PARTNERS DESCRIPTION AND RELEVANCE, COMPLEMENTARITY

For work on this scale one has to build an effective team; it cannot be done by an individual alone. I am fortunate to have an excellent group of colleagues to collaborate with, both at INRIA and abroad. For this research project I expect to continue my ongoing collaboration with Peter Sewell, employed by the Computer Laboratory of the University of Cambridge, and his team, in particular Jaroslav Sevcik (RA), Scott Owens (RA), Susmit Sarkar (RA), and Mark Batty (PhD student). Their experience with models of multiprocessor memory models, compilation and high-level languages is highly valuable for this project. Viktor Vafeiadis, previously at Cambridge, is now at MPI-SWS in Kaiserslautern, Germany. Vafeiadis expertise in the analysis of concurrent algorithms is complementary and will provide inspiration for all this work on compilation of high-level languages. Sewell and Vafeiadis have their own funding to work on the research presented in this proposal. Luc Maranget, chargé de recherche in the Moscova project-team at INRIA Paris-Rocquencourt (as myself), will contribute with his fundamental experience in building tools to validate the formalisation of hardware memory models.

5.2. QUALIFICATION DU COORDINATEUR DE LA PROPOSITION DE PROJET/ QUALIFICATION OF THE PROPOSAL COORDINATOR

The collaboration with Peter Sewell's group in Cambridge started in 2003 and is today well established. In the past we have worked on language design for distributed computation (Sewell et al., 2007) and tool support for semantics (Sewell et al., 2010). Our ongoing collaboration is now focused on formalisation of hardware weak-memory models (Sarkar et al., 2009; Sewell et al., 2010) and verified compilation of concurrent programming languages

(Sevcik et al., 2011). Funding this research proposal will enable me to pursue this fruitful collaboration.

5.3. QUALIFICATION, RÔLE ET IMPLICATION DES PARTICIPANTS / QUALIFICATION AND CONTRIBUTION OF EACH PARTNER

	Nom / Name	Prénom / First name	Emploi actuel / Position	Discipline / Field of research	Personne. mois* / PM	Rôle/Responsabilité dans la proposition de projet/ Contribution to the proposal 4 lignes max
Coordinateur/responsable	Zappa Nardelli	Francesco	CR, INRIA	Computer science	36	Coordinator and main investigator. Tasks 1-5.
Autres membres	Maranget	Luc	CR, INRIA	Computer science	12	Power/ARM memory model, tool-support. Tasks 2 and 5.
	Sewell	Peter	EPSRC Research Fellow, U. Cambridge, UK	Computer science	12	Design of programming languages, semantics, compilation, tool-support. Tasks 2,3 and Task 5.
	Vafeiadis	Viktor	MPI-SWS, Germany	Computer science	12	Algorithm verification, compilation, optimisations, semantics. Tasks 2-4.

* à renseigner par rapport à la durée totale du projet

6. JUSTIFICATION SCIENTIFIQUE DES MOYENS DEMANDES / SCIENTIFIC JUSTIFICATION OF REQUESTED RESSOURCES

This proposal is wide-ranging and challenging, aiming not just to develop reasoning techniques for relaxed memory execution (neglected in past research on concurrency verification), but to do so above semantic models that are faithful to the real-world processors and modern high-level languages. To do this requires an effective team with broad expertise. The project partners are all supported by their own grants to work on this project. I request funding for three man-years for PostDoc, funding for work visits to the project partners (which are vital to this project) and for travelling to attend conferences, and funding for workstations and laptops.

- *Équipement / Equipment.*

We request funds for two workstations for the PostDocs and project coordinator, and also one portable machine, for work and presentation while travelling (a total of 3 machines at 2000 € each).

- *Personnel / Staff*

We request funds for three man-years salary for PostDocs (each man-year costs 49k €). Each student will contribute to one of the main tasks that compose this research project.

Note that, even if members of foreign institutions, the contribution of Sewell and Vafeiadis have been taken into account in the computation of the number of person months

in document A (listed as "Personnel permanent"), and the corresponding costs are calculated on the DR2 scale (Sewell) and the CR1 scale (Vafeiadis).

- *Prestation de service externe / Subcontracting*

- *Missions / Travel*

Regular working visits to the project partners in Cambridge and Kaiserslautern are vital for this project. In addition I request funding to attend major relevant conferences where we hope to present our work (e.g. CAV, CONCUR, DAMP, ESOP, FM, ICFP, LICS, PLDI, PODC, POPL, PPoPP, SOSP, and TPHOLs), and for PostDoc student attendance at summer schools such as the International Summer School on Trends in Concurrency.

I request funds for 4 visits per year at 1000 € each for the project coordinator, 2 visits per year at 1000 € each for Maranget, and a total of 6 visits for the three PostDoc students at 1000 € each. I also request funds for brief trips to colleagues in France (2000 €). I request funds for two conference attendances or visits to industrial labs (particularly those of processor vendors) per year, at 1500 € each.

- *Dépenses justifiées sur une procédure de facturation interne / Costs justified by internal invoicies*

- *Autres dépenses de fonctionnement / Other expenses*

7. ANNEXES / ANNEXES

7.1. RÉFÉRENCES BIBLIOGRAPHIQUES / REFERENCES

S. V. Adve, K. Gharachorloo. *Shared memory consistency models: A tutorial*. Computer, 29(12):66–76, 1996.

A. Adir, H. Attiya, G. Shurek. *Information-flow models for shared memory with an application to the PowerPC architecture*. IEEE Trans. Parallel Distrib. Syst., 14(5):502–515, 2003.

J. Alglave. *A shared memory poetics*. Phd thesis, 2010.

J. Alglave, A. Fox, S. Ishtiaq, M. Myreen, S. Sarkar, P. Sewell, F. Zappa Nardelli. *The semantics of Power and ARM multiprocessor machine code*. In Proc. DAMP, 2009.

J. Alglave, L. Maranget, S. Sarkar, P. Sewell. *Fences in weak memory models*. In Proc. CAV, 2010.

D. Aspinall, J. Sevcik. *Formalising Java's data race free guarantee.* In Proc. TPHOLs, LNCS, 2007.

N. Benton, U. Zarfaty. *Formalizing and verifying semantic type soundness for a simple compiler*. In Proc. PPDP, 2007.

H.-J. Boehm. *A solid foundation for x86 shared memory: technical perspective.* CACM, 2010.

H.-J. Boehm. *Threads cannot be implemented as a library*. In Proc. PLDI, 2005.

H.-J. Boehm and S.V. Adve. *Foundations of the C++ concurrency memory model*. In Proc. PLDI, 2008.

P. Cenciarelli, A. Knapp, E. Sibilio. *The Java memory model: Operationally, denotationally, axiomatically.* In Proc. ESOP, 2007.

A. J. Chlipala. *A certified type-preserving compiler from lambda calculus to assembly language.* In Proc. PLDI, 2007.

J. Gosling, B. Joy, G. Steele, and G. Bracha. *Java(TM) Language Specification, The (3rd Edition) (Java Series), chapter Memory Model*, pages 557–573. Addison-Wesley Professional, July 2005.

A. Hobor, A. Appel, F. Zappa Nardelli. *Oracle semantics for concurrent separation logic*. In Proc. ESOP, 2008.

L. Lamport, *How to make a multiprocessor computer that correctly executes multiprocess programs.* IEEE Trans. Comput., C-28(9):690-691, 1979.

JTC1/SC22/WG14 — C. http://www.open-std.org/jtc1/sc22/wg14/.

X. Leroy. *A formally verified compiler back-end*. Journal of Automated Reasoning, 2009.

V. M. Luchangco. *Memory consistency models for high-performance distributed computing*. PhD thesis, MIT, 2001.

J. Manson, W. Pugh, and S. V. Adve. *The Java memory model*. In Proc. POPL. 2005.

P. McKenney, J. Walpole. *«What is RCU, fundamentally?»* Linux Weekly News. 2007. http://lwn.net/Articles/262464/.

P. McKenney, R. Silvera. *Example POWER Implementation for C/C++ Memory Model.* 2010. http://www.rdrop.com/users/paulmck/scalability/paper/N2745r.2010.02.19a.html.

S. Owens, S. Sarkar, P. Sewell. *A better x86 memory model: x86-TSO*. In Proc. TPHOLs, 2009.

S. Owens. *Reasoning about the implementation of concurrency abstractions on x86-TSO.* In Proc. ECOOP, 2010.

M. Parkinson, R. Bornat, P. O'Hearn. *Modular verification of a non-bocking stack.* In Proc. POPL, 2007.

W. Pugh. *The Java memory model is fatally flawed*. Concurrency - Practice and Experience, 12(6):445–455, 2000.

S. Sarkar, P. Sewell, F. Zappa Nardelli, S. Owens, T. Ridge, T. Braibant, M. Myreen, J. Alglave. *The semantics of x86-CC multiprocessor machine code*. In Proc. POPL, 2009.

J. Sevcik, D. Aspinall. *On validity of program transformations in the Java memory model*. In Proc. ECOOP, 2008.

J. Sevcik, V. Vafeiadis, F. Zappa Nardelli, S. Jagannathan, P. Sewell. *Relaxed-memory concurrency and verified compilation.* In Proc POPL, 2011 (to appear).

P. Sewell, J.J. Leifer, K. Wansbrough, F. Zappa Nardelli, M. Allen-Williams, P. Habouzit, V. Vafeiadis, *Acute: High-level programming language design for distributed computation*. J. Funct. Program. 17(4-5): 547-612 (2007).

P. Sewell, S. Sarkar, S. Owens, F. Zappa Nardelli, M. O. Myreen. *x86-TSO: A rigorous and usable programmer's model for x86 multiprocessors*. C. ACM, 53(7):89–97, 2010.

P. Sewell, F. Zappa Nardelli, S. Owens, G. Peskine, T. Ridge, S. Sarkar, R. Strnisa. *Ott: Effective tool support for the working semanticist*. J. Funct. Program. 20(1): 71-122 (2010)

A. Terekov. *Brief tentative example x86 implementation for C/C++ memory model.* 2008. http://www.decadent.org.uk/pipermail/cpp-threads/2008-December/001933.html.

V. Vafeiadis. *Proving correctness of highly-concurrent linearisable objects.* In Proc. PPoPP, 2006.

7.2. BIOGRAPHIES / CV, RESUME

Francesco ZAPPA NARDELLI *(34 - 25/12/1976 – http://moscova.inria.fr/~zappa)*

Chargé de recherche (CR1), Moscova research-team, INRIA Paris-Rocquencourt.

Professional Preparation

- Laurea (M.S.) in Computer Science, University of Pisa, Italy, October 2000.
- DEA Programmation (M.S.), University of Paris Sud, France, September 2000.
- Ph.D. in Computer Science, University of Paris 7, December 2003.

Appointments

- 2004–present, Research Scientist, INRIA Paris-Rocquencourt, France.
- 2003–2004, Post-doctoral grant, University of Cambridge, UK.
- 2002, Curie Research Fellow, Sussex University, UK.
- 2000–2003, PhD student, University of Paris 7, France.

Selected publications

- J. Sevcik, V. Vafeiadis, F. Zappa Nardelli, S. Jagannathan, P. Sewell. *Relaxed-memory concurrency and verified compilation*. In Proc POPL, 2011 (to appear).
- P. Sewell, S. Sarkar, S. Owens, F. Zappa Nardelli, M. Myreen. x86-*TSO: a rigorous and usable programmer's model for x86 multiprocessors.* Commun. ACM 53(7), 2010.
- T. Wrigstad, F. Zappa Nardelli, S. Lebresne, J. Ostlund, J. Vitek. *Intgrating typed and untyped code in a scripting language.* In Proc POPL, 2010.
- S. Sarkar, P. Sewell, F. Zappa Nardelli, S. Owens, T. Ridge, T. Braibant, M. Myreen, J. Alglave. *The semanctics of x86-CC multiprocessor machine code. elaxed-memory concurrency and verified compilation*. In Proc POPL, 2009.
- A. Hobor, A. Appel, F. Zappa Nardelli. *Oracle semantics for concurrent separation logic*. In Proc ESOP, 2008.

Zappa Nardelli is author of 14 papers in peer-reviewed international conferences and 5 papers published in journals (J. ACM, J. Funct. Program. (2), Inf. Comput, CACM).

Others

- Main author of Ott, a tool for large-scale semantics (http://moscova.inria.fr/~zappa/software/ott).
- Member of the CEA-EDF-INRIA summer school executive commitee.
- Member of the POPL 2012 program committe.

Luc MARANGET *(http://moscova.inria.fr/~maranget)*

Luc Maranget is Chargé de Recherche (CR1) in the Moscova research-team at INRIA Paris-Rocquencourt. He is an expert of compilation of functional languages and distributed applications. He is a core implementor of the OCaml compiler, and the main architect of the JoCaml system. Recently he developed efficient tools to explore the memory model of multiprocessor hardware, and carried on extensive testing of the Power memory model.

Peter SEWELL *(http://www.cl.cam.ac.uk/~pes20)*

Peter Sewell is a Reader and EPSRC Leadership Fellow at the University of Cambridge Computer Laboratory, following a Royal Society University Research Fellowship (1999-2007) and a PhD with Robin Milner (1995). His research aims to build rigorous foundations for the engineering of real-world computer systems, to make them better-understood, more robust, and more secure. He has published widely in leading venues in semantics, programming languages, networking, and security, and currently focusses on the relaxed memory models of multiprocessors and concurrent languages. This work is currently funded by two EPSRC grants (£2.1M), with project partners from IBM, ARM, Microsoft Research, and the Java/C++ concurrency community.

Viktor VAFEIDAIS *(http://www.mpi-sws.org/~viktor/)*

Viktor Vafeiadis ia an Independent Researcher at Max-Planck Institute for Software Systems (MPI-SWS), Germany, following post-doctoral positions at the University of Cambridge and Microsoft Research Cambridge, and undergraduate studies at the University of Cambridge. He is an expert on program analysis/verification, program logics, programming languages, and concurrency. He is author of publications in leading conferences (including POPL, CAV, ECOOP, VMCAI) and journals (JFP). He recently became interested in compiler verification.

7.3. IMPLICATION DES PERSONNES DANS D'AUTRES CONTRATS / STAFF INVOLVMENT IN OTHER CONTRACTS

As discussed above, Peter Sewell and Viktor Vafeiadis are supported by their own grants to work on the research object of this proposal; they are members of foreign research institutions and we do not report their grants in the table below.

Between January 2009 and December 2010 the collaboration between the project coordinator and Sewell was partially supported by the INRIA program "Équipes associées", which covers some of the travelling expenses between the two sites. This financial support has been extended to 2011, and consists in 8k € restricted to travelling expenses; man-months does not apply.

	Nom de la personne participant au projet / name	Personne. Mois / PM	Intitulé de l'appel à projets, source de financement, montant attribué / Project name, financing institution, grant allocated	Titre du projet : Project title	Nom du coordinateur / coordinator name	Date début & Date fin / Start and end dates
N°1	Zappa Nardelli, Maranget		INRIA Équipes associées, INRIA, 8k €	Équipes associées MM	Zappa Nardelli	January 2011 / December 2011