VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kansliets noteringar Kod 2014-45813-116186-15	Dnr 2014-5545

2014 Project Research Grant

Area of science
Natural and Engineering Sciences

Announced grants
Research grants NT April 9, 2014

Total amount for which applied (kSEK)

2015	2016	2017	2018	2019
930	930	962	1013	

APPLICANT

Name(Last name, First name) Clarke, Dave	**Date of birth** 710715-6452	**Gender** Male
Email address dave.clarke@it.uu.se	**Academic title** Associate professor	**Position** Lektor
Phone +46 18 - 471 4032	**Doctoral degree awarded (yyyy-mm-dd)** 2003-04-15	

WORKING ADDRESS

University/corresponding, Department, Section/Unit, Address, etc.
Uppsala universitet
Institutionen för informationsteknologi
Datalogi
Box 337
75105 Uppsala, Sweden

ADMINISTRATING ORGANISATION

Administrating Organisation
Uppsala universitet

DESCRIPTIVE DATA

Project title, Swedish (max 200 char)
SCADA: Skalbar Data för Parallellism Överallt

Project title, English (max 200 char)
SCADA: Scalable Data for Pervasive Parallelism

Abstract (max 1500 char)

Parallel computers are pervasive, and future computers of all scales will be provide their power through parallelism. Thus, all applications will need to be parallel to exploit the available computing resources. Doing so requires programming language support.

Future programming languages need to express both parallelism, to overcome sequential traps present in existing programming languages, and data distribution, to overcome memory hierarchy problems such as high latency, limited bandwidth, and contention, due to data being laid out poorly in memory.

Many research programming languages offer constructs for expressing parallelism, but support for flexible data distribution policies for all data types (including objects and algebraic data types) is lacking (even though extensive support is available for arrays and matrices).

The proposed research will extend the parallel-by-default programming language Encore with constructs for specifying data distribution and layout for all Encore data types. These constructs will allow data distribution information to be gradually incorporated into programs, supporting agile development approaches. These constructs will be implemented in the Encore compiler, and evaluated using both benchmarks and case studies.

The research will enable improved scalability and performance of existing applications, via a gradual tailoring of deployment concerns, permitting developers to optimise locally rather than globally.

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod	2014-45813-116186-15
Name of Applicant	Clarke, Dave
Date of birth	710715-6452

Abstract language
English

Keywords
parallelism, scalability, data

Review panel
NT-2

Project also includes other research area

Classification codes (SCB) in order of priority
10201, 10205,

Aspects

Continuation grant
Application concerns: New grant
Registration Number:

Application is also submitted to

similar to: **identical to:**

ANIMAL STUDIES

Animal studies
No animal experiments

OTHER CO-WORKER

Name(Last name, First name) ,	**University/corresponding, Department, Section/Unit, Addressetc.**
Date of birth	**Gender**
Academic title	**Doctoral degree awarded (yyyy-mm-dd)**

Name(Last name, First name) ,	**University/corresponding, Department, Section/Unit, Addressetc.**
Date of birth	**Gender**
Academic title	**Doctoral degree awarded (yyyy-mm-dd)**

Name(Last name, First name) ,	**University/corresponding, Department, Section/Unit, Addressetc.**
Date of birth	**Gender**
Academic title	**Doctoral degree awarded (yyyy-mm-dd)**

Name(Last name, First name) ,	**University/corresponding, Department, Section/Unit, Addressetc.**
Date of birth	**Gender**
Academic title	**Doctoral degree awarded (yyyy-mm-dd)**

VETENSKAPSRÅDET THE SWEDISH RESEARCH COUNCIL	Kod 2014-45813-116186-15
	Name of Applicant Clarke, Dave
	Date of birth 710715-6452

ENCLOSED APPENDICES

A, B, C, N, S

APPLIED FUNDING: THIS APPLICATION

Funding period (planned start and end date)

2015-01-01 -- 2018-12-31

Staff/ salaries (kSEK)

Main applicant	**% of full time in the project**	**2015**	**2016**	**2017**	**2018**	**2019**
David Clarke	25	266	272	279	286	
Other staff						
PhD Student	80	484	513	534	578	
	Total, salaries (kSEK):	750	785	813	864	

Other project related costs (kSEK)	**2015**	**2016**	**2017**	**2018**	**2019**
Computer costs	40	5	5	5	
Premises	30	30	30	30	
Travel	100	100	104	104	
Other (minor equipment, printing, books, etc)	10	10	10	10	
Total, other costs (kSEK):	180	145	149	149	

Total amount for which applied (kSEK)

2015	2016	2017	2018	2019
930	930	962	1013	

ALL FUNDING

Other VR-projects (granted and applied) by the applicant and co-workers, if applic. (kSEK)

Funds received by the applicant from other funding sources, incl ALF-grant (kSEK)

Funding source	**Total**	**Proj.period**	**Applied 2015**
KU Leuven GOA (Belgium)	2081	2010-2015	
Project title	**Applicant**		
Trusted Embedded Networked Systems	Bart Preneel, Ingrid Verbauwhede, Frank Piessens, Dave Clarke		

Funding source	**Total**	**Proj.period**	**Applied 2015**
EU FP7 OpenX	4367	2014-2017	
Project title	**Applicant**		
UpScale: From Inherent Concurrency to Massive Parallelism through Type-based Optimizations	Dave Clarke, Tobias Wrigstad (UU) and others		

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2014-45813-116186-15

Name of Applicant
Clarke, Dave

Date of birth
710715-6452

POPULAR SCIENCE DESCRIPTION

Popularscience heading and description (max 4500 char)

Skalbar Data för Parallellism Överallt

Parallellism är överallt och genomsyrar allt. Nära nog alla datorer är idag parallella i någon del. På grund av begränsningar i den teknik vi använder idag kommer denna trend att fortsätta och innebära ännu mer parallellism framgent.

En konsekvens av denna trend är att för att kunna utnyttja de resurser vi har tillgängliga måste vi skriva parallella program, i parallella programspråk. Det vi tidigare kallat parallellprogrammering, vikt åt en liten skara specialister främst inom beräkningsvetenskap, har nu blivit vardagsprogrammering. Alla program måste vara parallella program och alla programmerare måste vara parallellprogrammerare. Vidare måste programmerarna säkerställa att programmen håller god prestanda även då storleken på data och plattform ökar. Detta är målet med skalbarhet.

Programspråket Encore söker lösa precis detta problem, delvis genom att erbjuda parallellism som standarduppförande. Men att uttrycka parallellism är inte nog i sig, programmeraren måste även kunna uttrycka var i minnet data finns. Annars finns stor risk för prestandaproblem då data måste flyttas till rätt plats innan det kan behandlas (latency), begränsningar i hur mycket data som kan flyttas i taget (bandwidth), och att flera "trådar" försöker komma åt samma data samtidigt (contention).

Det finns en uppsjö programspråk i forskningsvärlden som erbjuder programmeraren att uttrycka parallellism, men saknar fullgott stöd för att beskriva exakt hur olika typer av data ska placeras i minnet. Detta innebär i grund och botten att de bara ger halva lösningen på skalbarhetsproblemet.

Den forskning vi föreslår kommer att utöka parallellism-som-standardprogramspråket Encore med konstruktioner för att specificera placering och layout för alla datatyper i Encore. Dessa konstruktioner kommer att medge gradvis specificering av var data ska placeras, och därmed ge programmeraren möjlighet att säga en del, vad hon tycker är viktigt, och låta kompilatorn räkna ut resten. Det passar väl in med dagens lättrörliga (agile) utvecklingsmetoder, och minskar den insats programmeraren måste göra tidigt i ett projekt för att nå skalbarhet.

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod	2014-45813-116186-15
Name of applicant	Clarke, Dave
Date of birth	710715-6452
Title of research programme	SCADA: Scalable Data for Pervasive Parallelism

Appendix A

Research programme

SCADA: Scalable Data for Pervasive Parallelism

Dave Clarke
Dept. of Information Technology, Uppsala University

Summary Parallelism is pervasive. Nowadays computing platforms of all scales, from smart phones to cloud computers, provide their power through parallel computing resources. Some smart phones even have 4 cores! The primary way computers will improve performance in the future will be by offering even higher degrees of parallelism, due to the so-called *power wall*.[1] The newly available computing power makes whole new classes of applications feasible, such as those in the big data realm, but it also means that all programmers must think about parallelism to ensure that their applications scale to take advantage of the computing resources available. In the past, increases in performance would automatically be exploited by programs, but now programs must be written with parallelism especially in mind: if a program is written in a scalable fashion, then additional parallel resources can readily be exploited, in particular as the size of data increases.

Many (research) programming languages, libraries and compilers overcome the first barrier to achieving scalability, namely so-called *sequential traps*—unnecessary sequential dependencies in the code force computations to run sequentially, thereby eliminating opportunities for parallelism.

Unfortunately, just expressing parallelism is not sufficient to achieve true scalability, due to the second barrier to scalability: the *memory hierarchy*. Memory is organised into a hierarchy consisting of small amounts of fast memory and larger amounts of slower memory (possibly in many levels, possibly on different machines). Problems memory hierarchy cause include (1) *latency* in accessing a memory location, (2) limited *bandwidth* when transferring data, and (2) *contention* when different threads access the same memory areas. In practice, all of these lead to a negative impact on performance. Although they do not decrease the amount of available parallelism, they can lead to execution times which are slower than in the sequential case—the essence of unscalability!

The way to overcome this second barrier is by specifying data distribution, which includes the layout and alignment of data in memory, its mapping to cores, and padding. High-level support for specifying aspects of data distribution for all data types available in common programming languages is lacking. Typically extensive support is available for arrays and matrices, but missing for objects and algebraic data types. Control over data distribution is crucial for achieving scalability—entire papers are published on scalable implementations of a single simple data structure. To force programmers to develop specialised data structures for all parts of their programs is too onerous. Instead, we need (compositional) data distribution declarations that are orthogonal to the semantics of the computation.

To overcome this second barrier, we will develop programming language technology for specifying aspects of the data distribution for a wide range of data types, and compiler transformations for modifying programs operating on these data types to work with the given data distribution. The language technology will be designed to allow the gradual incorporation of data distribution into programs, thereby supporting agile development methodologies. As a vehicle for exploring the language and the underlying compiler transformations, SCADA will build upon the parallel-by-default programming language Encore, which at present aims only to address the first of the two barriers above. The results will be evaluated using both small-scale benchmarking and larger-scale case studies, consisting of a variety of applications operating on different kinds of data.

[1]Increasing processor clock speeds results in more heat, but this can no longer be dissipated using air cooling.

The three core contributions of SCADA form a toolbox of comprehensive data distribution declarations, a gradual approach to using them, and compilation techniques that exploit them. This toolbox will be designed so that it can be equally exploited in a library or using design patterns, rather than just within a compiler. The results will enable improved scalability and performance of existing applications, supporting agile development methodologies that enable the late binding of crucial data distribution decisions, ultimately permitting developers to optimise locally rather than globally.

1 Purpose and Aims

The purpose of this project is to develop programming language technology to specify data distributions and compilation techniques to transform programs using them to support the writing of scalable parallel programs. Scalability ensures that data (of increasing size) can be mapped to all scales of computer platform, from small-scale multicore, through manycore, to cloud-based platforms. The techniques developed will make it easier for programmers to specify their data in a way that guarantees scalability, in particular, by allowing algorithmic details/business logic to be specified separately from the data distribution information required to scalability.

The first aim is to develop extensive and flexible language technology for expressing data distribution for an extensive range of data types. The data distribution declarations will be parameterised by application- and platform-specific information enabling programs to run with different sized data on different platforms.

The second aim is to develop techniques to support the gradual addition of such data distribution declarations to code. This support is compatible with modern agile development methodologies, and it will allow programs to be incrementally and locally enhanced with detailed data distributions. The gradual framework will support the data distribution constraints of different phases of computation and describe how data distributions change over the lifetime of a program.

The third aim is to develop new compiler transformations that transform user programs to exploit data distribution declarations (and to fill in the gaps when information has been omitted.)

The fourth aim is to implement the language technology in an existing compiler, for validation purposes. For this we have chosen the compiler for the Encore programming language.

The firth aim is to evaluate the results with small-scale micro-benchmarks and larger-scale case studies, to provide feedback on both the performance and usability of the programming language features.

2 Survey of the Field

A vast amount of research has been devoted to parallel computing, but as this field has traditionally worked to support scientific computing which reply heavily on array and matrix data types.

One common recent trend is for languages to incorporate higher-order combinators for expressing parallelism. Thread building blocks [25] and the Cilk(++) platform [3] support many parallel programming abstractions, integrated into inherently sequential languages like C and C++. Numerous Haskell libraries support parallel programming within a high-level programming language [31, 28]. These are designed to have good locality, but when this fails to happen there is no high-level programming language support to tune the data distribution. The Merge framework [27] offers an extensive library of MapReduce-style algorithmic skeletons, and the ability to dispatch to specific algorithms based on data size and architecture characteristics in a programmer-friendly fashion. Changing algorithm is useful for achieving scalability, but again the control of data distribution is missing. Triolet [34] optimises algorithmic skeletons to achieve good data locality, by using higher-level abstractions rather than loops. Unfortunately, the data types operated on are restricted to arrays.

Classical high-performance languages like HPF [24] offer a wide spectrum of data distribution directives. These were applicable only to array and matrix data types. HPF's coarrays [33] allows arrays be distributed across different 'images' (analogous to core or thread), and programmed in a high-level way, but again this is limited to arrays and matrices.

Modern high performance programming languages such as X10 [7], Chapel [6] and Fortress [23] offer a wide range of novel features for expressing parallel computations, including many higher-order operators like folds (reduction).

X10 has a conceptual global address space and the notion of place to support scalability, and operations for moving activities to places, forcing programmer to write code that operates only on local data. In addition, X10 offers a generalised concept of an array (called a rail), and builds upon these distributions that link arrays and places. These resemble coarrays in HPC, but are more abstract. Unfortunately, the distributions do not apply to all data types.

Fortress offers many parallel combinators and low-level directives for data placement and distributed are available, but again only for arrays.

Chapel adds more support for higher-order operations like scans and folds than predecessors like HFP, but do not go as far as adding data distribution support for arbitrary data types. Chapel has limited support for distribution in its global-local view splitting, which talks about computations across processors and computations on processors and how they combine. Chapel further supports separation of parallelism from locality specifications, but on a very low-level which requires programmers to roll their own data distribution and message passing, without support.

A number of techniques are available for transforming data types to be better suited to parallel computation. For instance, Parallel ML offers transformations of nested data parallelism suitable for multicore computers [2], improving upon earlier versions which were suitable only for vector processors. Nested data parallelism better suits higher-order operations on nested lists (and some encodings of graphs and trees), but it falls short of general data types and offers no control over data distribution. Murarasu et al. [30] present techniques for compressing (specific) data structures in order to exploit GPUs. These can be used as the basis for some of our more generic algorithms. SpiceC [20] achieve degrees of scalability using thread local data and copy-and-commit techniques for specifying the transfer of data between threads. This is similar to the PI's minimal ownership model [13] and its adaptation to thread-local data [37], and will be at the core of some techniques for object-level data distribution.

Numerous languages provide more advanced support for data distribution. The programming model underlying Sequoia [19] makes vertical aspects of the memory hierarchy available so that programmers can explicitly orchestrate data movement. The abstractions are based on a hierarchy of tasks. Originally, this applied only to regular data structures, but recent work [1] added primitives for communication with parent tasks and dynamically spawning child tasks, which allow irregular computations to be implemented. Dietz et al. [18] present an extensive collection of data declarations (grids, distributions, regions and arrays) for the ZPL language. Arrays are allocated over regions and data values are distributed according to some distribution over a grid of processors. Though extensive, and possible to express complex algorithms like adaptive mesh refinement, they are limited to arrays and matrices. The Ypnos [31] programming language provides detailed descriptions of the layout of grid-based computations in terms of stencils. This language is embedded in Haskell and uses advanced Haskell features to express properties of data. Again, properties are limited to arrays and matrices.

Other approaches aim to achieve good performance through so-called transactional memory [21]. As this is orthogonal to data distribution, it will not be considered in this project.

From an application perspective, popular big data application frameworks such as Hadoop,[2]

[2]http://hadoop.apache.org/

Storm[3] and Kafka[4] do not provide developers with special support for data distribution and layout, and they lack support gradual distribution. Instead applications must be (re)written drastically to support the abstractions provided by these frameworks.

3 Project Description The project will develop programming language technology for expressing data distribution in a scalable fashion and compilation algorithms for modifying code to use them. These will be formalised, implemented, and validated using both small scale micro-benchmarks and larger scale case studies. Initially programming language constructs for specifying data distribution will be specified (Activity A1: *Flexible Data Distribution*). Then, following the gradual typing approach, the notion of gradual data distribution will be developed to enable partial and incremental specification of data distribution (Activity A2: *Gradual Data Distribution*). Because data distribution in isolation cannot fully address scalability issues, the next activity considers how to compile parallel abstractions for a given data distribution (Activity A3: *Integration Data Distribution and Parallelism*). An ongoing activity will implement the features designed in Activities A1 and A2 within the Encore compiler (Activity A4: *Implementation*). Finally, there will be two validation activities. Case studies will be developed to explore the design space and to gain experience with the new declarations (Activity A5: *Case Studies*), and benchmarking will allow evaluation of the design space covered by the resulting implementation (Activity A6: *Profiling and Benchmarking*).

3.1 Flexible Data Distribution (A1) This activity focusses on providing means to express data distribution for a wide range of data types, such as those provided by the Encore language. Encore is an object-oriented language based on active objects, but it also has support for polymorphic algebraic data types (as in Haskell, for instance), low-level array and matrix types and embedded structs (as in C, in contrast to Java), abstract types representing aspects of parallel computations (such as futures), and ownership and uniqueness typing — all fully compositional. The data distribution constructs will specify how data is laid out in memory and spread across cores or machines, and may include alignment and padding information. Specifications will be declarative and have no behavioural content — the meaning of a computation should is not affected by data distribution declarations.

This activity will consider only complete data distributions. Issues such as data distribution inference and data redistribution are handled in Activity A2.

Design Space Survey The first phase will survey data distribution constructs from the literature. These mostly apply to arrays and matrices, and in that space can be quite rich. Other policies that will be added to the survey will include: encapsulating fields of an object within the object vs. bunching fields of all objects of a given type together (related to the array of structs vs. struct of arrays layout choice); storing objects near their creator; storing objects near their owners; storing objects of the same type together; padding objects to avoid false sharing on a single node; packed allocation strategies of linked data structures; and for tree-shaped data structures, possibilities such as allocating one branch locally and another branch further away. Orthogonal concerns and how they are treated will also be enumerated, such as provide guarantees to align data to cache line boundaries. In addition, we will record the various trade-offs that need to be made when choosing a data distribution. For example, avoiding false sharing by padding trades off space efficiency for parallel performance, but in other cases a more compact data representation would be preferable. The resulting data distributions developed in SCADA will enable a single user-defined data to support multiple distributions, even within a single program, to cater for both of these important use cases.

[3] `http://hortonworks.com/hadoop/storm/`
[4] `https://kafka.apache.org`

Lightweight Models The second phase will consist of building a compositional calculus of data descriptions to support a wide spectrum of data types. The semantics of the data distribution calculus will be given in a very lightweight fashion, essentially describing the mapping of data residing in an abstract address space to the concrete $(node \times address)$-space (along with size, alignment and padding information, when required). This will provide the basis for code transformations for each data distribution. These code transformations, considered in Activity A3, will form the core of the implementation scheme used in the compiler.

Higher-order and User-defined Abstractions The final phase will investigate higher-order abstractions specifying relationships between various data and introduce parameterised , user-defined data distribution declarations.

Higher-order abstractions will specify co-locality, such as that two fields of an object should be co-located or that the contents of those fields should be co-located, and alignment of data, such as arrays. Ultimately, these will capture that such elements are likely accessed at that same time or in similar patterns and hence should be co-located and aligned. Similar declarations will express padding of objects to avoid false sharing and other kinds of contention for a particular resource, ultimately to avoid co-location.

Finally, support will be added for user-defined and parameterised data distribution abstractions. These will be realised as a restricted constraint language over relevant parameters such as (expected) data size and features of the platform's architecture (size of memory, structure of hierarchy, number of nodes, ...). True scalability will only be achieved if the data distribution can change as the size of data and characteristics of the architecture change.

3.2 Gradual Data Distribution (A2)

The assumption of Activity A1 is that the data distribution of all data is fully specified. This onerous upfront requirement works against the agile development methodology we are envisaging. To counter this, Activity A2 will develop this gradual framework for data distribution declarations. The goal is to design programming language technology—language constructs, supporting theory and implementation techniques, to allow the partial specification of data distribution. In our setting, partial data declarations essentially correspond to a constraint on the space of data declarations, and the dynamic type checks of gradual typing correspond to redistribution operations. With these changes, gradual type checking of Siek [36] and others can be adapted to enable partial data distribution specifications and infer places where data redistribution will need to occur.

Now it is well-known that (parallel) programs proceed in a number of phases and that the distribution of data used in one part of a program may not be optimal in later parts of the program. In such situations, there are a number of options to consider: a costly redistribution step, running the one of the parts of the computation with a non-optimal distribution, or choosing some intermediate distribution that is not locally optimal but may be globally so. Now gradual typing can track changes to data distribution through the program, but to reason about such choices in a gradual typing setting, we will introduce a cost calculus for estimating the cost of each possibility, to guide programmers in choosing the best one—though ultimately, benchmarking will be the only way to decide which is best. The flexible data distribution declarations with added costs, and support from constraints generated by the gradual typing approach, will allow programmers to more easily explore the trade-offs between different data distribution options.

3.3 Integrating Data Distribution and Parallelism (A3)

Neither parallelism nor data distribution can be considered in isolation. For instance, memory contention of co-scheduled computations can kill potential gains in speed-up , and can even result in a slowdown. (Indeed, memory contention so drastically affects scalability, the *only* data distribution directive offered by Java describes contention.)

To address this issue, the goal of this activity is develop a toolbox compilation techniques for transforming various styles of parallelism to match the given data distribution. The compilation techniques will ensure that tasks are appropriately scheduled, possibly employing dynamic scheduling techniques such as work stealing, to better exploit parallel resources.

Data Parallelism The lowest level consists of ensuring that data in arrays and matrices are packed together and similar operations on the elements of these can be performed in parallel, so that GPUs and other fine-grained parallel computing units can be exploited. To achieve this, this activity will adapt existing research on nested data parallelism [2] and embedding data parallelism into a higher-order language [28] to interact with the data distribution specifications. The constraints on data distributions that permit such transformations will be determined, based on the underlying limitations of the GPUs etc, and incorporated into the cost calculus of gradual data distributions.

Parallelism within Objects Two requirements need to be satisfied to allow methods to run in parallel within the same object: the methods must not modify the same fields (to avoid data races) and the fields that they do access must be stored on separate cache lines (to avoid memory contention). The former property is guaranteed by known typing techniques. To address the second requirement, different means for storing object fields will be developed, including packing them all together and packing them in a scattered fashion with appropriate padding. Compilation algorithms will be developed to transform method bodies for the new data distribution.

Parallelism above Objects Encore offers a wide range of naturally parallel, higher-order constructs for operating at the level above objects, including LINQ-like queries [29] across objects, involving operations such as filters, maps and reduction, higher-order function as in Haskell and orchestration combinators as in Orc [26]. These encompass functionality provided by MapReduce frameworks [17].

The research will develop compilation techniques for transforming these higher-order constructs to operate using the given data distribution. Furthermore, we will use gradual deployment descriptions of Activity A2 to express the ideal data distribution of such computations and the distributions of their intermediate results.

The result of this phase will expand the power of popular MapReduce frameworks with better built-in language support for controlling data distribution.

Fork-Join Parallelism A very effective means for improving performance is to use fork-join concurrency.[5] This activity will consider the interaction between data distribution and fork-join concurrency. In this case, the notion of lenses [4], which provide modifiable views on a data structure, can be adapted to allow the safe execution on parts of a data structure without having to copy or move the data structure, in a way that guarantees that different threads do not access the same part of memory. These *data distribution-sensitive lenses* will enable the compiler, for instance, to ensure that splitting of data occurs appropriately at memory boundaries to avoid contention between parallel threads.

3.4 Implementation (A4)

The Encore compiler will be used as a testbed for implementing and experimenting with the language features designed in Activities A1–A3.

The Encore compiler generates C and LLVM code for the Pony backend [15], which is a high-performance framework for actor- and active object-based parallelism, with a built-in concurrent garbage collector and an actor-local memory allocation strategy. Changes will be required to most compilation phases, to support well-formedness checking, transformation of code based on data distribution declarations, and

[5]Doug Lea, author of the Java concurrency libraries, says that fork-join is the most effective way of achieving improved performance in most cases. (Personal communication.)

backend code generation. Additional run-time support for various data distribution policies will be added to the Pony backend, in particular, to integrate appropriately with the memory manager—for instance, bitmaps may be require to record which parts of a packed collection of object fields are valid. The backend will rely heavily on generic programming, generating code based on the structure of both data types and data distribution declarations, based on a few core functions. Appropriate pragmas will be inserted into the resulting C/LLVM code to ensure that the lower-level C/LLVM compiler eliminates any overhead introduced by this technique.

The Distributed/Cloud version will target Distributed Pony, which is also being developed in the UpScale project as part of the backend infrastructure to support Encore. Distributed Pony will provide support for aspects of distributed programming such as reliable messaging and so forth, allowing the SCADA to focus exclusively on data distribution.

3.5 Case Studies (A5) Case studies will provide validation of the research from the programmer experience and expressiveness perspectives. The case studies will be carried out in three phases: at the start of the project, in the middle, and towards end. The case study will be drawn from three different application domains: *computational social science*, *biochemical modelling* and *crowd simulation*, each of which involves a large amount of data processing on different core data structures: (social) graphs, (phylogenetic) trees, and (neighbourhood) grids, respectively.

In general, we will be porting/adapting code developed by others, rather than developing applications ourselves.

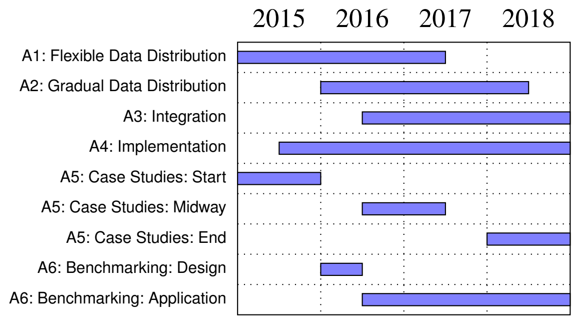
In the start, case studies will be implemented in a number of 'competing' programming languages to understand in pragmatic detail what these offer and what the weaknesses of their underlying approaches are.

Midway case studies will be reimplemented in early versions of Encore to exploit the data distribution declarations developed up until this point to provide an preliminary evaluation of their usability/expressiveness and of the performance of the compiler. The role of this case study is primarily *formative* in nature, to help guide research in the final part of the project.

Towards the end, case studies will be reimplemented in final versions of Encore to exploit the sophisticated versions of the data distribution declarations that will be developed. The role of this case study is primarily *summative*, to evaluate the resulting language technology (and to suggest directions for future research).

3.6 Benchmarking (A6) While the case studies focus on full applications, benchmarking activity will provide micro-benchmarks to understand the space of data distribution declarations. A benchmarking suite will be developed by analysing the case studies and other applications and extracting their core computational patterns, and by consulting the relevant literature. The benchmarking suite will then be systematically applied throughout the project, and extended periodically, as new functionality is developed. The results of the benchmarking will pinpoint which the kinds of patterns that yield good scalability and also where implementation improvements need to be made.

Planning Activities A1–A6 are scheduled as follows:



The PI will focus mostly on the language design aspects and design of the benchmarking suites and case studies. The PhD student will work initially on the implementation aspects, and then on language design issues after his/her background knowledge and experience increase. The PhD student will also implement and maintain the benchmark suite and case studies.

4 Significance

Parallelism is pervasive in modern computing platforms, and, due to the *power wall*, increasing the amount of parallelism will be the main way of increasing processor performance in the future. Consequently, in order to scale, software—*all software*—needs to be written to expose as much parallelism as possible to exploit the available computing power.

The current state of the art of parallelism in software development is to let a few experts deal with such aspects of a program. This is no longer possible, and in a near future, *all programmers must become parallel programmers*.

Most existing programming languages were designed for sequential execution, with parallel programming support added later. This means that they have built-in sequential traps that destroy potential speedup. Furthermore, automatic parallelisation techniques have reached their limits, and moreover never applied to general-purpose programs anyway. Specialised parallel programming languages fail to work well for general-purpose applications. And no parallel programming language addresses the scalability of all data types.

Parallel-by-default programming languages such as Encore are required to avoid the sequential traps, but they need to offer extensive means for expressing data distribution to achieve true scalability. For a single program to scale across the entire spectrum of available computing platforms requires suitably flexible deployment descriptions. Providing such deployment descriptions is the main goal of this project.

The gradual approach to deployment will better support agile development practices, allowing developers to incrementally specify precise data distributions for core parts of their application to get the performance they desire, while leaving other less performance-crucial parts untouched. This will make the results more readily applicable.

Overall the project will contribute programming language technology that supports the development of scalable programs in important areas like data science, but also for general-purpose applications. The results will apply to potentially all programs running on potentially all platforms, enabling programmers to chose the degree of performance tuning they need.

5 Preliminary Results

Development of the Encore compiler is underway and by the start of the planned work, all the core parallel programming constructs will be implemented. In particular, the Pony backend is already fully operational and quite robust, and the Distributed Pony backend is reaching stability.

Recent work by the PI on active-object based programming languages [9, 8], alias control in this context [13], and semantics and verification of programs using futures [16], has informed many of the design decisions in the Encore language to pave the way for the proposed research. The specification of data distribution for objects will upon the PI's work on ownership types [14, 11]; the PI has also developed a gradual version of ownership types [35].

Recent work by the PI on improving the scalability of the Reo coordination language on multicore platforms [12, 32], has provided practical experience in programming such systems.

The proposed use of generic programming techniques in SCADA builds on the PI's earlier work on Generic Haskell [10] and on applying shape theory in parallel programming [22].

6 International Collaboration

The PI has a number of ongoing international collaborations related to the proposed research. The most direct is with members of the UpScale project, specifically, with Sophia Drossopoulou and Nobuko Yoshida (Imperial College) on aspects of type system design for expressing the structure and behaviour of computation, with Sylvan Clebsch (Imperial College) on the Pony back end, and with Frank de Boer (CWI) and Einar Broch Johnsen (Oslo University) on active object programming languages, deployment in cloud platforms and application-specific scheduling.

Ongoing work on developing the Reo coordination language for both multicore and distributed architectures with Farhad Arbab (CWI) and José Proença (KU Leuven) will provide further expertise on developing systems for such platforms. Furthermore, the PIs maintains active contact with his former group DistriNet (KU Leuven), who also have expertise in cloud computing.

7 Independent Line of Research This proposal is independent of any other proposal submitted to the Swedish Research Council.

This project in relates to the EU FP7 OpenX project UpScale, for which the PI is co-site leader. The current proposal takes a complementary angle to UpScale, focussing on data distribution instead of parallelism. The proposed research will build upon the results of UpScale, which will have been running for 1 year when this project would start.

This project will be embedded within the UPMARC group in the Department of Information Technology at Uppsala University. This group explores a diverse range of topics associated with parallel computing, and, as such, provides invaluable expertise in related areas, spanning low-level computer architecture, the compilation of parallel languages (specifically, Erlang), and alias control. For instance, the computer architecture group have extensive experience measuring the negative effects of the memory hierarchy, though they lack the expertise in high level language design. The Erlang team focusses on implementing the Erlang virtual machine. As Erlang has no means for expressing data distribution, the work proposed here is orthogonal. The members of UPMARC working on alias control are co-partners on the UpScale project, where they focus on capability type-based optimisations.

References

[1] M. Bauer, J. Clark, E. Schkufza, and A. Aiken. Programming the memory hierarchy revisited: supporting irregular parallelism in Sequoia. In Cascaval and Yew [5], pages 13–24.

[2] L. Bergstrom, M. Fluet, M. Rainey, J. H. Reppy, S. Rosen, and A. Shaw. Data-only flattening for nested data parallelism. In A. Nicolau, X. Shen, S. P. Amarasinghe, and R. W. Vuduc, editors, *PPOPP*, pages 81–92. ACM, 2013.

[3] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou. Cilk: An efficient multithreaded runtime system. In J. Ferrante, D. A. Padua, and R. L. Wexelblat, editors, *PPOPP*, pages 207–216. ACM, 1995.

[4] A. Bohannon, J. N. Foster, B. C. Pierce, A. Pilkiewicz, and A. Schmitt. Boomerang: resourceful lenses for string data. In G. C. Necula and P. Wadler, editors, *POPL*, pages 407–419. ACM, 2008.

[5] C. Cascaval and P.-C. Yew, editors. *Proceedings of the 16th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP 2011, San Antonio, TX, USA, February 12-16, 2011*. ACM, 2011.

[6] B. L. Chamberlain, D. Callahan, and H. P. Zima. Parallel programmability and the Chapel language. *IJHPCA*, 21(3):291–312, 2007.

[7] P. Charles, C. Grothoff, V. A. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, and V. Sarkar. X10: an object-oriented approach to non-uniform cluster computing. In R. E. Johnson and R. P. Gabriel, editors, *OOPSLA*, pages 519–538. ACM, 2005.

[8] D. Clarke, N. Diakov, R. Hähnle, E. B. Johnsen, I. Schaefer, J. Schäfer, R. Schlatte, and P. Y. H. Wong. Modeling spatial and temporal variability with the HATS abstract behavioral modeling language. In M. Bernardo and V. Issarny, editors, *SFM*, volume 6659 of *Lecture Notes in Computer Science*, pages 417–457. Springer, 2011.

[9] D. Clarke, E. B. Johnsen, and O. Owe. Concurrent objects à la carte. In D. Dams, U. Hannemann, and M. Steffen, editors, *Concurrency, Compositionality, and Correctness*, volume 5930 of *Lecture Notes in Computer Science*, pages 185–206. Springer, 2010.

[10] D. Clarke and A. Löh. Generic Haskell, specifically. In J. Gibbons and J. Jeuring, editors, *Generic Programming*, volume 243 of *IFIP Conference Proceedings*, pages 21–47. Kluwer, 2002.

[11] D. Clarke, J. Östlund, I. Sergey, and T. Wrigstad. Ownership types: A survey. In D. Clarke, J. Noble, and T. Wrigstad, editors, *Aliasing in Object-Oriented Programming*, volume 7850 of *Lecture Notes in Computer Science*, pages 15–58. Springer, 2013.

[12] D. Clarke and J. Proença. Partial connector colouring. In M. Sirjani, editor, *COORDINATION*, volume 7274 of *Lecture Notes in Computer Science*, pages 59–73. Springer, 2012.

[13] D. Clarke, T. Wrigstad, J. Östlund, and E. B. Johnsen. Minimal ownership for active objects. In G. Ramalingam, editor, *APLAS*, volume 5356 of *Lecture Notes in Computer Science*, pages 139–154. Springer, 2008.

[14] D. G. Clarke, J. Potter, and J. Noble. Ownership types for flexible alias protection. In B. N. Freeman-Benson and C. Chambers, editors, *OOPSLA*, pages 48–64. ACM, 1998.

[15] S. Clebsch and S. Drossopoulou. Fully concurrent garbage collection of actors on many-core machines. In A. L. Hosking, P. T. Eugster, and C. V. Lopes, editors, *OOPSLA*, pages 553–570. ACM, 2013.

[16] F. S. de Boer, D. Clarke, and E. B. Johnsen. A complete guide to the future. In R. De Nicola, editor, *ESOP*, volume 4421 of *Lecture Notes in Computer Science*, pages 316–330. Springer, 2007.

[17] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150. USENIX Association, 2004.

[18] S. J. Deitz, B. L. Chamberlain, and L. Snyder. Abstractions for dynamic data distribution. In *HIPS*, pages 42–51. IEEE Computer Society, 2004.

[19] K. Fatahalian, D. R. Horn, T. J. Knight, L. Leem, M. Houston, J. Y. Park, M. Erez, M. Ren, A. Aiken, W. J. Dally, and P. Hanrahan. Sequoia: programming the memory hierarchy. In *SC*, page 83. ACM Press, 2006.

[20] M. Feng, R. Gupta, and Y. Hu. SpiceC: scalable parallelism via implicit copying and explicit commit. In Cascaval and Yew [5], pages 69–80.

[21] M. Herlihy and J. E. B. Moss. Transactional memory: Architectural support for lock-free data structures. In A. J. Smith, editor, *ISCA*, pages 289–300. ACM, 1993.

[22] C. B. Jay, D. Clarke, and J. Edwards. Exploiting shape in parallel programming. In *IEEE Second International Conference on Algorithms and Architectures for Parallel Processing*, pages 295–302, 1996.

[23] G. L. S. Jr., E. E. Allen, D. Chase, C. H. Flood, V. Luchangco, J.-W. Maessen, and S. Ryu. Fortress (Sun HPCS language). In D. A. Padua, editor, *Encyclopedia of Parallel Computing*, pages 718–735. Springer, 2011.

[24] K. Kennedy, C. Koelbel, and H. P. Zima. The rise and fall of High Performance Fortran. *Commun. ACM*, 54(11):74–82, 2011.

[25] W. Kim and M. Voss. Multicore desktop programming with Intel Threading Building Blocks. *IEEE Software*, 28(1):23–31, 2011.

[26] D. Kitchin, A. Quark, W. R. Cook, and J. Misra. The Orc programming language. In D. Lee, A. Lopes, and A. Poetzsch-Heffter, editors, *FMOODS/FORTE*, volume 5522 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 2009.

[27] M. D. Linderman, J. D. Collins, H. Wang, and T. H. Y. Meng. Merge: a programming model for heterogeneous multi-core systems. In S. J. Eggers and J. R. Larus, editors, *ASPLOS*, pages 287–296. ACM, 2008.

[28] T. L. McDonell, M. M. T. Chakravarty, G. Keller, and B. Lippmeier. Optimising purely functional GPU programs. In G. Morrisett and T. Uustalu, editors, *ICFP*, pages 49–60. ACM, 2013.

[29] E. Meijer, B. Beckman, and G. M. Bierman. LINQ: reconciling object, relations and XML in the .NET framework. In S. Chaudhuri, V. Hristidis, and N. Polyzotis, editors, *SIGMOD Conference*, page 706. ACM, 2006.

[30] A. F. Murarasu, J. Weidendorfer, G. Buse, D. Butnaru, and D. Pflüger. Compact data structure and scalable algorithms for the sparse grid technique. In Cascaval and Yew [5], pages 25–34.

[31] D. A. Orchard and A. Mycroft. Efficient and correct stencil computation via pattern matching and static typing. In O. Danvy and C. chieh Shan, editors, *DSL*, volume 66 of *EPTCS*, pages 68–92, 2011.

[32] J. Proença and D. Clarke. Interactive interaction constraints. In R. De Nicola and C. Julien, editors, *COORDINATION*, volume 7890 of *Lecture Notes in Computer Science*, pages 211–225. Springer, 2013.

[33] J. Reid and R. W. Numrich. Co-arrays in the next Fortran standard. *Scientific Programming*, 15(1):9–26, 2007.

[34] C. I. Rodrigues, T. B. Jablin, A. Dakkak, and W. mei W. Hwu. Triolet: a programming system that unifies algorithmic skeleton interfaces for high-performance cluster computing. In J. Moreira and J. R. Larus, editors, *PPOPP*, pages 247–258. ACM, 2014.

[35] I. Sergey and D. Clarke. Gradual ownership types. In H. Seidl, editor, *ESOP*, volume 7211 of *Lecture Notes in Computer Science*, pages 579–599. Springer, 2012.

[36] J. G. Siek and W. Taha. Gradual typing for objects. In E. Ernst, editor, *ECOOP*, volume 4609 of *Lecture Notes in Computer Science*, pages 2–27. Springer, 2007.

[37] T. Wrigstad, F. Pizlo, F. Meawad, L. Zhao, and J. Vitek. Loci: Simple thread-locality for java. In S. Drossopoulou, editor, *ECOOP*, volume 5653 of *Lecture Notes in Computer Science*, pages 445–469. Springer, 2009.

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2014-45813-116186-15

Name of applicant
Clarke, Dave

Date of birth
710715-6452

Title of research programme
SCADA: Scalable Data for Pervasive Parallelism

Appendix B

Curriculum vitae

Appendix B: CV — Dave Clarke

Name: David (Dave) Gerard Clarke   Date-of-birth: 15th July, 1971   Nationality: Australian

Academic Degrees

PhD, (Computer Science), University of New South Wales, Sydney, Australia. 2003.

Bachelor of Science (First Class Honours), Australian National University, Australia. 1994.

Employment

September 2013–Present *Associate Professor.* Department of Information Technology, Uppsala University, Sweden.

2008–Present *Assistant professor.* (10% since September 2013). DistriNet Research Group, Department of Computer Science, KU Leuven, Belgium.

2004–2008 *Postdoctoral Researcher.* SEN3 (Coordination Languages) Group, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands.

2001–2004 *Postdoctoral Researcher.* Software Technology Group, Department of Computer Science, Utrecht University, Utrecht, The Netherlands.

Post-doctoral Researcher Supervision

José Proença. KU Leuven, Belgium. 2010–present.

PhD Supervision

Past (as principal advisor/promotor): Radu Muschevici, KU Leuven (November 2013), Dimiter Milushev, KU Leuven (June 2013), Ilya Sergey, KU Leuven (November 2012), Tobias Wrigstad, KTH, Sweden (May 2006). Plus 5 as copromotor.

Current (as promotor): Marco Patrignani. KU Leuven (Started October 2010), Rula Sayaf, KU Leuven (Started March 2011), Adriaan Larmuseau, Uppsala University (Started September 2012). Plus 4 as copromotor.

Research Funding Obtained (Since 2009)

HATS: Highly Adaptable and Trustworthy Software using Formal Methods. (Site leader) FP7 FET. 412,825.60 EUR. 1 March 2009–31 February 2013.

COGNAC: Coordination and Ownership in Graphs of Networked Actors. (Principle Investigator). Flemish FWO research project grant. Joint project with the Vrij Universiteit Brussel. 271,912 EUR. 1 January 2011–31 December 2014

SPION: Security and Privacy in Online Social Networks. (Co-applicant). Flemish IWT Strategic Basic Research Fund (SBO). 1 January 2011–31 December 2014. 493,667 EUR.

DesignerTypeLab: Hybrid Enforcement and Certification of Domain Specific Program Annotations (Sole Investigator). KU Leuven Basic Research Funds (BOF/START). 250,000 EUR. 1 October 2009–30 September 2013.

TENSE: Trustworthy Networked Embedded Systems. (Co-applicant). KU Leuven Concentrated Research Actions (GOA). 1 October 2010–30 September 2015. 1,250,000 EUR (shared among 4 partners).

UpScale: From Inherent Concurrency to Massive Parallelism through Type-based Optimizations. (Co-site leader). FP7 FET OpenX project. 1 March, 2014–31 February, 2017. 428,800 EUR

International Awards

ACM Most influential paper of *Conference on Object-Oriented Programming Languages and Systems* (OOPSLA) 1998. Awarded at OOPSLA 2008 for the paper *Ownership Types for Flexible Alias Protection*. David Clarke, John Potter, James Noble.

Most influential paper of *Australian Software Engineering Conference* (ASWEC) 1998. Awarded at ASWEC 2008 for the paper *The Ins and Outs of Objects*. John Potter, James Noble, David Clarke.

Professional Duties (Excerpt)

Scientific Advisory Board of European FP7 Project *Envisage*.

PC chair or Co-chair of COORDINATION 2010 Conference and FMPSPLE 2013 Workshop, IWACO 2003 Workshop.

Doctoral Symposium chair at SPLC 2014.

Member of Steering Committee of COORDINATION Conference and Chair of Steering Committee of IWACO workshop series.

Member of 20 Conference Programme Committees since 2005 (including OOPSLA 2005, ESOP 2008, ESOP 2012, ECOOP 2014, ECOOP 2015). Member of 23 Workshop Program Committees since 2003.

Reviewer for *Instituut voor Wetenschappelijk Onderzoek en Innovatie (IWOIB)* , Belgium 2011, *Technologiestichting STW*, NL, 2008, *German-Israeli Foundation for Scientific Development (GIF)*, 2006 and 2010,

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod	2014-45813-116186-15
Name of applicant	Clarke, Dave
Date of birth	710715-6452
Title of research programme	SCADA: Scalable Data for Pervasive Parallelism

Appendix C

List of publications

Appendix C.
Publication List — Dave Clarke

Note that peer-reviewed conference proceedings are the primary form of publication in Computer Science.

Citations are based on Google Scholar (2 April, 2014).
Most relevant articles are marked with ⋆.

Theses

[1] Dave Clarke. (2002). Object Ownership & Containment. PhD thesis. *University of New South Wales*.
Number of citations: 204

Journal Special Issues as Editor

[2] Dave Clarke, Gul A. Agha (Eds.). (2013). Special Issue devoted to selected articles from Coordination Models and Languages (Coordination 2010). *Science of Computer Programming* 78(2).

[3] Ina Schaefer, Rick Rabiser, Dave Clarke (Eds.). (2013). Special Issue devoted to Diversity. *International Journal on Software Tools for Technology Transfer* 14(5).

Books as Editor

[4] Dave Clarke, James Noble, Tobias Wrigstad (Eds.). (2013). Aliasing in Object-Oriented Programming: Types, Analysis and Verification. *Lecture Notes in Computer Science, State-of-the Art Survey* volume 7850. Springer.

[5] Dave Clarke, Gul A. Agha (Eds.). (2010). Coordination Models and Languages, 12th International Conference, COORDINATION 2010, Amsterdam, The Netherlands, June 7-9, 2010. Proceedings. *Lecture Notes in Computer Science* volume 6116. Springer.

International Journals (Refereed)

[6] Sung-Shik T.Q. Jongmans, Dave Clarke, and José Proença. (Accepted). A Procedure for Splitting Data-Aware Processes and its Application to Coordination. *Science of Computer Programming*.

[7] Thomas Heyman, Riccardo Scandariato, Dave Clarke, Wouter Joosen. (Accepted). Compositional security verification of software architectures. *Software and Systems Modeling*.

[8] Dave Clarke, Michiel Helvensteijn, Ina Schaefer. (Accepted). Abstract delta modeling. *Mathematical Structures in Computer Science*.

[9] Dave Clarke, Gul A. Agha. (2013). Preface to the special issue on Coordination Models and Languages (Coordination 2010). *Science of Computer Programming*. 78(2). pp 45–146.

[10] Ina Schaefer, Rick Rabiser, Dave Clarke, Lorenzo Bettini, David Benavides, Goetz Botterweck, Animesh Pathak, Salvador Trujillo, Karina Villela. (2012) Software Diversity: state of the art and perspectives. *International Journal on Software Tools for Technology Transfer*. 14(5). pp 477–495.
Number of citations: 18

[11] Marcello Bonsangue, Dave Clarke, Alexandra Silva, A. (2012). A model of context-dependent component connectors. *Science of Computer Programming*, 77(6): 685–706.
Number of citations: 5

[12] Ilya Sergey, Dave Clarke. (2012). A correspondence between type checking via reduction and type checking via evaluation. *Information Processing Letters*, 112(1–2), 13–20.
Number of citations: 5

[13] Dave Clarke, José Proença, Alexander Lazovik, Farhad Arbab. (2011). Channel-based coordination via constraint satisfaction. *Science of Computer Programming*, 76(8), 681–710.
Number of citations: 20

[14] Mohammad Izadi, Marcello Bonsangue, Dave Clarke. (2011). Büchi automata for modeling component connectors. *Software and Systems Modeling*, 10(2), 183–200.
Number of citations: 4

[15] Dave Clarke. (2008) A Basic Logic for Reasoning about Connector Reconfiguration. *Fundamentae Informatica* 82(4), 361–390.
Number of citations: 19

[16] Tobias Wrigstad, Dave Clarke. (2007) Existential Owners for Ownership Types. *Journal of Object Technology* 6(4), 141–159.
Number of citations: 30

[17] Dave Clarke, David Costa, Farhad Arbab. (2007). Connector Colouring I: Synchronization and Context Dependency. *Science of Computer Programming* 66(3), 205–225.
Number of citations: 83

[18] Alex Potanin, James Noble, Dave Clarke, Robert Biddle. (2006). Featherweight Generic Confinement. *Journal of Functional Programming* 16(6), 793–811.
Number of citations: 33

Book Chapters

[19] Marko van Dooren, Dave Clarke. Bart Jacobs. (2013). Subobject-oriented Programming. *Formal Methods for Components and Objects*. *Lecture Notes in Computer Science* volume 7866. Springer. pp 38–82.
Number of citations: 3

[20] Dave Clarke, James Noble, Tobias Wrigstad. (2013). Beyond the Geneva Convention on the Treatment of Object Aliasing. In *Aliasing in Object-Oriented Programming: Types, Analysis and Verification. Lecture Notes in Computer Science, State-of-the Art Survey* volume 7850. Springer. pp 1–6.
Number of citations: 0

⋆ [21] Dave Clarke, Johan Östlund, Ilya Sergey, Tobias Wrigstad. (2013). Ownership Types: A Survey. Beyond the Geneva Convention on the Treatment of Object Aliasing. In *Aliasing in Object-Oriented Programming: Types, Analysis and Verification. Lecture Notes in Computer Science, State-of-the Art Survey* volume 7850. Springer. pp 15–58.
Number of citations: 8

[22] Rula Sayaf, Dave Clarke. (2013). Access Control Models for Online Social Networks. In *Social Network Engineering for Secure Web Data and Services*. ICI-Global. pp 32–65.
Number of citations: 3

[23] Dave Clarke, Nikolay Diakov, Reiner Hähnle, Einar Broch Johnsen, Ina Schaefer, Jan Schäfer, Rudolf Schlatte, Peter Y. H. Wong. (2011). Modeling spatial and temporal variability with the HATS abstract behavioral modeling language. In Bernardo M., Issarny V. (Eds.), *Formal Methods for Eternal Networked Software Systems*. Lecture Notes in Computer Science volume 6659, 417–457. Springer.
Number of citations: 27

International Conference Publications (Refereed)

[24] Marco Patrignani and Dave Clarke (2014). Fully abstract trace semantics for low-level protection mechanisms. In *Proceedings of the 20014 ACM symposium on Applied Computing (SAC)*, pp 1369–1373. ACM.
Number of citations: 0

[25] Marco Patrignani, Dave Clarke and Frank Piessens. (2013). Secure Compilation of Object-Oriented Components to Protected Module Architectures. *Proceedings of 11th Asian Symposium on Programming Languages and Systems*. Lecture Notes in Computer Science volume 8301. pp 176–191.
Number of citations: 2

[26] Dimiter Milushev and Dave Clarke. (2013). Incremental hyperproperty model checking via games. *Proceedings of the 18th Nordic Conference on Secure IT Systems*. Lecture Notes in Computer Science volume 8208. pp 247–262.
Number of citations: 2

[27] José Proença and Dave Clarke. (2013). Interactive Interaction Constraints. *15th International Conference on Coordination Models and Languages.* Lecture Notes in Computer Science volume 7890. Springer. pp 211–225.
Number of citations: 1

[28] Ilya Sergey, Dominique Devriese, Matthew Might, Jan Midtgaard, David Darais, Dave Clarke and Frank Piessens. (2013). Monadic Abstract Interpreters. *34th annual ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI)*. pp 399–410.
Number of citations: 1

[29] Dominique Devriese, Ilya Sergey, Dave Clarke, Frank Piessens (2013). Fixing idioms: a recursion primitive for applicative DSLs. *Proceedings of the ACM SIGPLAN 2013 Workshop on Partial Evaluation and Program Manipulation (PEPM)*. pp 97–106. ACM.
Number of citations: 2

[30] Dimiter Milushev, Dave Clarke. (2012). Coinductive Unwinding of Security-Relevant Hyperproperties. *Secure IT Systems - 17th Nordic Conference, NordSec*. Lecture Notes in Computer Science volume 7617, pp 121–136.
Number of citations: 2

[31] Michael Lienhardt, Dave Clarke. (2012). Conflict Detection in Delta-Oriented Programming. *5th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. Lecture Notes in Computer Science volume 7609, pp 178–192.
Number of citations: 4

[32] Ilya Sergey, Jan Midtgaard, Dave Clarke. (2012). Calculating graph algorithms for dominance and shortest path. *Mathematics of Program Construction.* Lecture Notes in Computer Science 7342, pp 132–156. Springer-Verlag.
Number of citations: 0

[33] Dimiter Milushev, Wim Beck and Dave Clarke. (2012). Noninterference via Symbolic Execution. In *FMOODS/FORTE*. Lecture Notes in Computer Science 7273, pp 152–168. Springer-Verlag.
Number of citations: 1

⋆ [34] Marko van Dooren and Dave Clarke. (2012). Subobject Transactional Memory. In *Coordination Models and Languages.* Lecture Notes in Computer Science 7274, pp 44–58. Springer-Verlag.
Number of citations: 1

⋆ [35] Dave Clarke, José Proença. (2012). Partial Connector Colouring. In *Coordination Models and Languages.* Lecture Notes in Computer Science volume 7274, pp 59–73. Springer-Verlag.
Number of citations: 9

[36] Dimiter Milushev, Dave Clarke. (2012). Towards incrementalization of holistic hyperproperties. In *Conference on Principles of Security and Trust.* Lecture Notes in Computer Science volume 7215, pp 329–348. Springer.
Number of citations: 5

⋆ [37] Ilya Sergey, Dave Clarke. (2012). Gradual Ownership Types. In Seidl, H. (Ed.). *Programming Languages and Systems (ESOP),* Lecture Notes in Computer Science volume 7211, pp 579–599. Springer.
Number of citations: 8

[38] Michael Lienhardt, Dave Clarke. (2012). Row types for delta-oriented programming. In: *ACM Conference Proceedings Series. Variability Modelling of Software-Intensive Systems (VaMoS)*, pp 121–128.
Number of citations: 4

[39] José Proença, Dave Clarke, Erik de Vink, Farhad Arbab. (2012). Dreams: a framework for distributed synchronous coordination. In *ACM Symposium on Applied Computing*, pp 1510–1515. ACM Press.
Number of citations: 9

[40] Radu Muschevici, José Proenca, Dave Clarke. (2011). Modular modelling of software product lines with feature nets. In Barthe, G., Pardo, A., Schneider, G. (Eds.) *9th international conference on software engineering and formal methods (SEFM)*, Lecture Notes in Computer Science 7041, pp 318–333. Springer-Verlag.
Number of citations: 9

[41] Marco Patrignani, Dave Clarke, Davide Sangiorgi. (2011). Ownership types for the join calculus. In: Bruni, R., Dingel, J. (Eds.). *Formal Techniques for Distributed Systems - Joint 13th IFIP WG 6.1 International Conference, FMOODS 2011, and 31st IFIP WG 6.1 International Conference, FORTE 2011*. Lecture Notes in Computer Science 6722, 289–303. Springer.

Number of citations: 1

[42] Dave Clarke, Radu Muschevici, José Proença, Ina Schaefer, Rudolf Schlatte. (2010). Variability modelling in the ABS language. In Aichernig, B., de Boer, F., Bonsangue, M. (Eds.) *9th international symposium on formal methods for components and objects (FMCO)*, Lecture Notes in Computer Science 6957, 204–224. Springer-Verlag.

Number of citations: 15

[43] Dave Clarke, Michiel Helvensteijn, Ina Schaefer. (2010). Abstract delta modeling. In *Proceedings of the ninth international conference on Generative programming and component engineering (GPCE)*. 13–22. ACM Press.

Number of citations: 54

⋆ [44] Dave Clarke, Einar Broch Johnsen, Olaf Owe. (2010). Concurrent objects à la carte. In Dams, D., Hannemann, U., Steffen, M. (Eds.) *Concurrency, Compositionality, and Correctness Essays in Honor of Willem-Paul de Roever*. Lecture Notes in Computer Science 5930, 185–206. Springer.

Number of citations: 6

[45] Farhad Arbab, Roberto Bruni, Dave Clarke, Ivan Lanese, Ugo Montanari. (2009). Tiles for Reo. In *Workshop on Algebraic Development Techniques*. Lecture Notes in Computer Science 5486, 37–55. Springer.

Number of citations: 25

[46] Marcello Bonsangue, Dave Clarke, D., Silva, A. (2009). Automata for context-dependent connectors. In *Coordination Languages and Models (COORDINATION)*. Lecture Notes in Computer Science 5521, 184–203. Springer.

Number of citations: 39

[47] Christian Koehler, Dave Clarke. (2009). Decomposing port automata. In *Proceedings of the 2009 ACM symposium on Applied Computing (SAC)*, 1369–1373. ACM.

Number of citations: 21

[48] Dave Clarke, Tobias Wrigstad, Johan Östlund, Einar Broch Johnsen. (2008) Minimal Ownership for Active Objects. *The Sixth ASIAN Symposium on Programming Languages and Systems* (APLAS). Lecture Notes in Computer Science 5356, 139–154. Springer.

Number of citations: 40

[49] Mohammad Izadi, Marcello M Bonsangue, Dave Clarke. (2008) Modeling Component Connectors: Synchronisation and Context-Dependency. In *Proceedings of 6th IEEE International Conferences on Software Engineering and Formal Methods* (SEFM). IEEE Computer Society.
Number of citations: 16

[50] Dave Clarke. (2008). Coordination: Reo, nets, and logic. In *Proceedings of Formal Methods for Components and Objects (FMCO)*. Lecture Notes in Computer Science 5382. 226–256. Springer.
Number of citations: 12

[51] Johan Östlund, Tobias Wrigstad, Dave Clarke, Beatrice Åkerblom. (2008). Ownership, Uniqueness and Immutability. *Objects, Components, Models and Patterns* (TOOL Europe) Lecture Notes in Business Information Processing 11, 178–197, Springer.
Number of citations: 38

⋆ [52] Frank de Boer, Dave Clarke, Einar Broch Johnsen. (2007). A Complete Guide to the Future. *European Symposium on Programming*, Lecture Notes in Computer Science 4421, 316–330. Springer.
Number of citations: 111

[53] Dave Clarke, Sophia Drossopoulou, James Noble, Tobias Wrigstad. (2007). Tribe: A Simple Virtual Class Calculus. *Aspect-Oriented Software Development (AOSD)*, ACM International Conference Proceeding Series 208, 121–134. ACM Press.
Number of citations: 46

[54] Alex Potanin, James Noble, Dave Clarke, Robert Biddle. (2006). Generic Ownership for Generic Java. In William Cook (Ed.) *ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA)*. ACM SIGPLAN Notices 41(10), 311–324. ACM Press.
Number of citations: 90

[55] Sophia Drossopoulou, Dave Clarke, James Noble. (2006) Type for Hierarchic Shapes (Summary). In Peter Sestoft (Ed.) *Programming Languages and Systems, 15th European Symposium on Programming (ESOP)*. Lecture Notes in Computer Science 3924, 1–6. Springer-Verlag.
Number of citations: 8

Highly Cited Earlier Works — International Conference Publications (Refereed)

[56] Dave Clarke, Tobias Wrigstad. (2003). External Uniqueness is Unique Enough. In Luca Cardelli (Ed.) *European Conference on Object-Oriented Programming (ECOOP*

'03). Lecture Notes in Computer Science 2743, 176–200. Springer.
Number of citations: 146

[57] Dave Clarke, Sophia Drossopoulou. (2002). Ownership, Encapsulation, and the Disjointness of Type and Effect. In *ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'02)*. ACM SIGPLAN Notices 37(11), 292–310, ACM Press.
Number of citations: 292

[58] Dave Clarke, James Noble, John Potter. (2001). Simple Ownership Types for Object Containment. In Jorgen Lindskov Knudsen (Ed.) *European Conference on Object-Oriented Programming (ECOOP'01)*. Lecture Notes in Computer Science 2072, 53–76, Springer-Verlag.
Number of citations: 137

[59] Dave Clarke, John Potter, James Noble. (1998). Ownership Types for Flexible Alias Protection. In *Proceedings of ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'98)*, *ACM SIGPLAN Notices* 33(10), 48–64, ACM Press.
Number of citations: 587

Published Workshop Reports

[60] Dave Clarke, Ina Schaefer, Maurice H. ter Beek, Sven Apel, Joanne M. Atlee. (2013). Formal methods and analysis in software product line engineering: 4th edition of FMSPLE workshop series. *SPLC 2013*. ACM pp 266–267.

[61] Dave Clarke, Sophia Drossopoulou, Peter Müller, James Noble, Tobias Wrigstad. (2008). Aliasing, confinement, and ownership in object-oriented programming. In *ECOOP Workshops*. Lecture Notes in Computer Science, 5475, 30-41. Springer.

[62] John Boyland, Dave Clarke, Gary Leavens, Francesco Logozzo, Arnd Poetzsch-Heffter. (2007). Formal Techniques for Java-Like Programs. In *ECOOP Workshops*. Lecture Notes in Computer Science, 4906, 99-107. Springer.

[63] Dave Clarke, Sophia Drossopoulou, James Noble, Tobias Wrigstad. (2007). Aliasing, confinement, and ownership in object-oriented programming. In *ECOOP Workshops*. Lecture Notes in Computer Science, 4906, 40-49. Springer.

Published Workshop Papers (Refereed)

[64] Radu Muschevici, Dave Clarke and José Proença. (2013). Executable modelling of dynamic software product lines in the ABS language. *FOSD@GPCE 2013*.

pp 17-24. ACM.
Number of citations: 0

[65] José Proença and Dave Clarke. (2013). Data abstraction in coordination constraints. *Advances in Service-Oriented and Cloud Computing – Workshops of ESOCC 2013,* Revised Selected Papers. Communications in Computer and Information Science. pp 159–173. Springer 2013.
Number of citations: 0

[66] Bo Gao, Bettina Berendt, Dave Clarke, Ralf De Wolf, Thomas Peetz, Jo Pierson, and Rula Sayaf. (2012). Interactive Grouping of Friends in OSN: Towards Online Context Management. *2012 IEEE 12th International Conference on Data Mining Workshops (ICDMW)* pp 555–562.
Number of citations: 3

[67] Marco Patrignani, Dave Clarke. (2013). Fully abstract trace semantics of low-level protection mechanisms. *Proceedings of the 24th Nordic Workshop on Programming Theory*, pp 43–45.
Number of citations: 0

[68] Sung-Shik T.Q. Jongmans, Dave Clarke, and José Proença. (2012). A Procedure for Splitting Processes and its Application to Coordination. *Proceedings 11th International Workshop on Foundations of Coordination Languages and Self Adaptation (FOCLASA)*. EPTCS 91. pp 79–96.
Number of citations: 8

[69] Joakim Bjørk, Dave Clarke, Einar Broch Johnsen, Olaf Owe. (2012). A Type-Safe Model of Adaptive Object Groups. *Proceedings 11th International Workshop on Foundations of Coordination Languages and Self Adaptation (FOCLASA)*. EPTCS 91. pp 1–15.
Number of citations: 0

[70] Marco Patrignani, Nelson Matthys, José Proença, Danny Hughes, Dave Clarke. (2012). Formal analysis of policies in wireless sensor network applications. *Third International Workshop on Software Engineering for Sensor Network Applications (SESENA)*, pp 15–21. IEEE.
Number of citations: 4

[71] Ilya Sergey, Dave Clarke. (2011). From type checking by recursive descent to type checking with an abstract machine. In *Language Descriptions, Tools and Applications* (LDTA), 1–7. ACM.
Number of citations: 3

[72] José Proença, Dave Clarke, Erik de Vink, Farhad Arbab. (2011). Decoupled execution of synchronous coordination models via behavioural automata. In *Proceedings of the International Workshop on the Foundations of Coordination Languages*

and Software Architecture (FOCLASA). Electronic Proceedings in Theoretical Computer Science 58, 65–79.
Number of citations: 8

[73] Dave Clarke, José Proença. (2010). Towards a theory of views for feature models . In *Proceedings of the First Intl. Workshop on Formal Methods in Software Product Line Engineering (FMSPLE 2010)*. Jeju Island, South Korea. Workshop Proceedings (Volume 2 : Workshops, Industrial Track, Doctoral Symposium, Demonstrations and Tools). Lancaster University.
Number of citations: 9

[74] Dave Clarke, Nikolay Diakov, Reiner Hähnle, Einar Broch Johnsen, Germán Puebla, Balthasar Weitzel, Peter Y. H. Wong. (2010). HATS - A formal software product line engineering methodology. In *Proceedings of the First Intl. Workshop on Formal Methods in Software Product Line Engineering (FMSPLE 2010)*. Jeju Island, South Korea. Workshop Proceedings (Volume 2: Workshops, Industrial Track, Doctoral Symposium, Demonstrations and Tools). Lancaster University.
Number of citations: 10

[75] Radu Muschevici, Dave Clarke, José Proença. (2010). Feature Petri Nets. In *Proceedings of the First Intl. Workshop on Formal Methods in Software Product Line Engineering (FMSPLE 2010)*. Jeju Island, South Korea. Workshop Proceedings (Volume 2: Workshops, Industrial Track, Doctoral Symposium, Demonstrations and Tools). Lancaster University.
Number of citations: 17

[76] Dave Clarke, Ilya Sergey. (2009). A semantics for context-oriented programming with layers. In *International Workshop on Context-Oriented Programming (COP '09)*, 7–13. ACM.
Number of citations: 15

[77] Dave Clarke, Pascal Costanza, Éric Tanter. (2009). How should context-escaping closures proceed? In *International Workshop on Context-Oriented Programming (COP '09)*. 1–6. ACM.
Number of citations: 10

[78] Dave Clarke, José Proença. (2009). Coordination via interaction constraints I: Local logic. In Bonchi, F., Grohmann, D., Spoletini, P., Tuosto, E. (Eds.), *Interaction and Concurrency Experience: Structured Interactions (ICE)*. Electronic Proceedings in Theoretical Computer Science 12, 17-39.
Number of citations: 4

[79] Dave Clarke, José Proença, Alexander Lazovik, Farhad Arbab. (2009). Deconstructing Reo. *Electronic Notes in Theoretical Computer Science* 229(2), 43-58.
Number of citations: 21

[80] José Proença, Dave Clarke. (2007). Coordination Models Orc and Reo Compared. In Carlos Canal, Pascal Poizat and Mirko Viroli (Eds.) *Proceedings of the 6th International Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA 2007)*. Electronic Notes in Theoretical Computer Science 194(4), 57–76. Elsevier.
Number of citations: 8

Informally Published Workshop Presentations (Refereed)

[81] Tobias Wrigstad, Dave Clarke. (2011). Is the world ready for ownership types? Is ownership types ready for the world?. In *International Workshop on Aliasing, Confinement and Ownership. IWACO '11.* Lancaster, UK, 25 July 2011.
Number of citations: 2

[82] Dave Clarke, Sophia Drossopoulou, James Noble. (2011). The roles of owners. In *International Workshop on Aliasing, Confinement and Ownership. IWACO '11*. Lancaster, UK, 25 July 2011.
Number of citations: 1

[83] Ilya Sergey, Dave Clarke. (2011). Towards Gradual Ownership Types. In *International Workshop on Aliasing, Confinement and Ownership. IWACO '11*. Lancaster, UK, 25 July 2011.
Number of citations: 3

[84] David Clarke. The Coordination Game: Find the Black Box. *International Workshop on Synthesis and Analysis of Component Connectors (SYANCO)* at ESEC/FSE 2007, Dubrovnik, Croatia, September 2007.
Number of citations: 0

[85] Johan Östlund, Tobias Wrigstad, Dave Clarke, Beatrice Åkerblom. (2007). Ownership, Uniqueness and Immutability. *International Workshop on Aliasing, Confinement and Ownership (IWACO)* at ECOOP, Berlin, Germany.
Number of citations: 0

Misc: Scientific Communication

[86] Dave Clarke, David Eppstein, Kaveh Ghasemloo, Lev Reyzin, András Salamon, Peter Shor, Aaron Sterling, Suresh Venkatasubramanian. (2010). Questions answered. In theory.: `http://cstheory.stackexchange.com/`. In *SIGACT (ACM Special Interest Group on Automata and Computability Theory) News*, 41(4), 58-60.

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2014-45813-116186-15
Name of applicant
Clarke, Dave
Date of birth
710715-6452

Title of research programme

Appendix N

Budget and research resources

Budget Justification for SCADA: Scalable Data for Pervasive Parallelism

Dave Clarke
Dept. of Information Technology, Uppsala University

1 Motivation

We ask for funding for one PhD student, plus 25% of the PI's time, which we believe is appropriate for running the project successfully from start to finish. A single dedicated PhD student, together with the PI should be able to perform the tasks outlined in the project description. The PI is requesting four years of funding due to the amount of infrastructure the project will build up. This framework will form the basis for the PI's future research.

Computer Costs We are applying for 40 KKR for computer costs, covering a laptop and/or desktop machine, display, etc. for the student, and some external hard drives for keeping backups. We expect to spend another 5 KKR each year on repairs, software, etc.

Premises The 30 KKR in the budget is 25% of the costs of the PI's room and the costs for a shared room for a PhD student.

Travel The field of computer science is driven by conference publication which necessitates travel. The money asked for covers 3–4 trips per year. A trip to an international conference costs approximately 20–30 KKR including registration and overheads. Sending the students to at least one summer school, such as the Oregon Programming Languages Summer School, early on, is desirable, both for educational and networking purposes.

Masters Students In departmental tradition, most students do their master projects externally at some company. While this is good for many reasons, it makes it hard to compete for the best talent as companies can offer monetary compensation. To this end, we would like to be able to offer one funded master project per year, where the salary would be nominal and paid on successful completion. We are asking for a mere 20 KKR/year for this, including overhead.

Other Having a 10 KKR "slush fund" for minor, unexpected costs is standard practise, to cover printing costs, buying books, etc.

2 Available Resources

The project will be conducted at Uppsala University by the PI and a doctoral student, with expected contributions from masters students. The two PhD students working on the EU FP7 OpenX project UpScale will contribute to the Encore framework upon which the research is being based.

We strive to recruit internationally in order to maintain a high-class working environment. The PI has experience (at his former institution) in recruiting internationally, which he plans to exploit to attract top-class PhD students.

VR is funding 100% of this project.

VETENSKAPSRÅDET THE SWEDISH RESEARCH COUNCIL	Kod 2014-45813-116186-15	Dnr 2014 -
	Name of applicant Clarke, Dave	
	Date of birth 710715-6452	Reg date 2014-04-09 21:20:25

Project title

SCADA: Scalable Data for Pervasive Parallelism

NT

NT-2

Research grants NT April 9, 2014
Project Research Grant
*Övrigt generellt
Uppsala universitet

Appendix S

Signatures

2015	2016	2017	2018	2019	
930	930	962	1013		2015-01-01 -- 2018-12-31

Applicant	**Date**	
Head of department at host University	**Clarifi cation of signature**	**Telephone**

A signature on the application is required not only from the applicant but also from the authorised representative of the administrating organisation/institution or equivalent (normally the head of the department or establishment where the research is to be conducted). The signature confirms that the department can accommodate the proposed research, position or equipment; that the costing in the application is approved for the department's part, that any proposed experimentation on human or animal subjects has been reported, and that the applicant has reported any secondary occupations and commercial ties (s)he may have, and nothing inconsistent with good research practice has thereby emerged. The applicant must have discussed these conditions with the representative of the administrating organisation/institution or equivalent before the latter approves and signs the application.

NOTE the exceptions!
Please note that a different procedure applies for the following types of grants:

- Infrastructure- Operation Grants
- Infrastructure- Grants for Expensive Equipment
- Infrastructure- Grants for Large Databases
- Framework Grant SIMSAM
- International Recruitment

For these types of grants the vice-chancellor, instead of the head of department, must sign Appendix S. Thereby (s)he confirms the commitments of the administrative organization. The details are specified in the Call for Proposals, published at www.vr.se.

Vetenskapsrådets noteringar
Kod

2014-45813-116186-15