Communicating Process Architectures 2013
P.H. Welch et al. (Eds.)
Open Channel Publishing Ltd., 2013

A Personal Perspective on the State of HPC in 2013

Christopher C.R. JONES
BAE Systems, Preston, England.
chris.c.jones@baesystems.com

Abstract. This paper is fundamentally a personal perspective on the sad state of High Performance Computing (HPC, or what was known once as Supercomputing). It arises from the author's current experience in trying to find computing technology that will allow codes to run faster: codes that have been painstakingly adapted to efficient performance on parallel computing technologies since around 1990, and have allowed effective 10-fold increases in computing performance at 5 year HPC up-grade intervals, but for which the latest high-count multi-core processor options fail to deliver improvement. The presently available processors may as well not have the majority of their cores as to use them actually slows the code – hard-won budget must be squandered on cores that will not contribute. The present situation is not satisfactory: there are very many reasons why we need computational response, not merely throughput. There are a host of cases where we need a large, complex simulation to run in a very short time. A simplistic calculation based on the nominal performance of some of the big machines with vast numbers of cores would lead one to believe that such rapid computation would be possible. The nature of the machines and the programming paradigms, however, remove this possibility. Some of the ways in which the computer science community could mitigate the hardware shortfalls are discussed, with a few more off the wall ideas about where greater compute performance might be found.

Keywords. HPC, electromagnetic simulations, non-scalability, programmability, crisis

Introduction

The point of parallel computing from an HPC (High Performance Computing) perspective is to be able to solve bigger problems faster than can be achieved with a single processor – this is about memory size and computing performance. It would be nice to be able to treat the combination of parallel computer and parallel code as the ultimate in object orientation – to add a new feature to the solver capability would be to add a new object. Sadly, this is not how it works in practice. Despite superficial resemblances, networks of objects are not a good match for networks of computers. Objects can interact in ways that are far more subtle and complex than message passing over wires, even with our best (software engineering) efforts to ensure low coupling. The result is almost the opposite of what is needed - to add a new feature requires new, additional code to be integrated into the existing object code under the current paradigm. Code maintenance and development, therefore, is seriously non-trivial - its cost depends on the size of the whole system, not the size of the new feature. Furthermore, obtaining good parallel computing performance and scalability to large numbers of cores is very hard to achieve in the first place, and yet harder to maintain through cycles of code development.

Hard as code performance is to get on large parallel machines, the hardware is becoming finer grained and intrinsically less homogeneous as the HPC industry slavishly tries to keep

up with what it thinks Moore's law was, claiming to supply us with ever more processing power via more and more cores on single silicon substrates. This is creating new problems:

1. Memory addressing for cores on a processor is different from that for separate processors. In practice, this is even more complicated as will be described briefly later.
2. Often a different parallel approach and library is used for core to core communications to that for processor to processor – e.g. OpenMP for the first and MPI for the second
3. Bandwidth and latency for communications is similarly different for the various layers.

This adds to the existing problem for the user of large parallel computers, that more processors and cores does not necessarily make a given problem run faster, and in general speed of solutions does not scale with number of cores – we cannot run a problem on a bigger computer and expect it to run as much faster (if at all) as we might naively expect. So a solution that might take 250 hours on 100 processors in the old currency will not run in 25 hours on 1000 cores or 2.5 hours on 10,000 cores, etc. In fact, we may find that without completely rewriting the solver, and possibly rethinking the numerical approach, we cannot speed up an existing problem at all. We might be able to run bigger problems, but not speed up the existing range of problems. While being able to run bigger problems is important, we also need to speed up the solution time. Yet, because of the interaction between processing time and the amount of data to be communicated between parallel processes, and the fact the individual cores are not contributing significantly to the continuing (apparent) increase in processor performance, each problem size for a particular solver has a *sweet spot* in core count where the performance is optimal, and above and below this core count the performance drops off. We may be criticised for following a data-parallel route to parallelising our solvers, but remember the earlier point – one of the driving reasons for using parallel computing is the access to more memory – this is not merely nice to have, it is absolutely essential.

It is therefore, the combination of the microprocessor designers' failing to comprehend the real issues of multi-core, i.e. parallel processing, and the failure of the computer scientists to offer a better programming paradigm that we can use that has led us to this position.

The present parallel computer hardware approach is wrong, but there are few options, so we just have to make it work. There are three major hardware issues to overcome:

1. the inhomogeneity of the communications which makes it harder to get real performance and requires special programming,
2. the contention between many cores trying to gain access to a single block of memory with limited memory access bandwidth, and
3. the lack of reliable, high bandwidth, low latency, homogeneous interconnect.

Hand-in-hand with the failure of HPC platforms to deliver realisable computing performance, knowledge in computer science (CSP [1,2,3,4,5], the π-calculus [6], occam [7, 8,9,10,11], etc.) has overtly not been adopted to attempt to mitigate the deficiencies in the hardware – or failed equally spectacularly to be disseminated to users.

A real source of frustration is that we did have technology, both hardware and software in the Transputer and occam, that with development over the intervening years since Inmos' demise would/could have avoided these issues. Development of programming potential has continued without a *"good"* hardware platform for implementation, and ignored by the HPC industry at large. Good science has continued in a few special places, but funding sources have not recognised the dangers lurking ahead and have failed to put sufficient priority and resources in those special places. Now, the writing is on the wall for scalable general high performance computing unless and until these issues are recognised and serious investment can be focused here.

1. Our Need for High Performance Computing (HPC)

HPC is increasingly being used in the Aerospace industry. Initially this has been in the risk reduction of the design function to remove ineffective solutions and optimise towards a best compromise. Aerodynamics and structural analysis were probably the areas where earliest HPC effort was directed. In the late 1980s and early 90s, the HPC platform of choice was the CRAY XMP followed by the YMP. These were multi-purpose, multi-user systems, set up for very many, relatively small jobs, while electromagnetic (EM) simulations were large jobs requiring the whole machine for periods of ten days and more at a time. This disjoint in job pattern resulted in only one partial EM simulation ever being accomplished at that time. From the early 1990s, it became feasible to do realistic electromagnetic simulations of lightning strikes to aircraft and their effect on coupling into aircraft systems cables. This became feasible at that time because of the subject-dedicated affordable parallel computing offered by Transputer technology followed by their emulation using consumer microprocessors.

One hundred and ninety two T800 processors provided equivalent performance to that achievable from the Cray YMP, but as a machine dedicated to electromagnetics, and with a single user provided the required access to long run times. With one more foray into shared supercomputing resources which failed, we have maintained EM resources since those early 1990s with increases in compute performance of about a factor of 10 at each upgrade at 5 yearly intervals. It is a sad but true fact that these increments in performance of the computing platform have been largely absorbed by more refinement in the representation of our aircraft, in improvements in the modelling of features and materials, and improvements in the code accuracy through better boundary conditions and switching from single to double precision. Hence, while our present computer is about 10,000 times faster than the first, the same simulation now takes 3 days instead of 10, but with significantly improved accuracy. The new technology and ideas growing from CSP and Transputer had opened up new opportunities on which we were able to capitalise. But the HPC Industry were quite happy to let people think everything was carrying on along the upward trend of ever-increasing computing power, even though the achievable performance was rising more slowly or was static. We have now got to a point where the technologies available actually cannot solve some large problems because of lack of scaling.

Developments in-hand at the present will introduce increased requirements as we move from Cartesian *"sugar cube"* discretisation of the problem and the space around it, to unstructured, body-conforming cell models. Another development is to model the generation of voltage arcs and thermal sparks when the very high lightning currents flow through joints in carbon fibre composite (CFC) structures. This modelling work would increase the compute time up to thousands of days without some acceleration techniques[1], but even so may take from 10 to 20 days per run on the next 10-fold upgrade in HPC power.

Installed antenna performance (IAP) modelling and electrostatic discharge (ESD) have become, and high intensity radiated fields (HIRF) simulations are also becoming, part of the virtual armoury with comparative problem sizes and run times, but which add to the overall amount of computing required. See Table 1.

Table 1. Typical problem sizes in EM on current HPC.

Simulation	Cells	Time-Steps	Run Time
Lightning & HIRF	150M to 600M	1M – 4M	1 – 4 days
ESD	4.6B	160k	4 – 5 days
IAP	<12B	<60k	1 day

[1] An adaptation of Holland's method [12] for reducing the value of the speed of light to reduce the number of time steps required to cover an elapsed electromagnetic interaction time.

It should be noted that HPC procurements for this EM simulation purpose are always done against a compute performance requirement, not a number of cores.

2. Opportunities that cannot be tapped

At present, the turn-around time for just the computing element of the simulations we do (for instance) for lightning, when a significant airframe or system design change is proposed (e.g. one that might change the performance of the lightning protection), is between 1 and 3 months[2]: that is, there is a solid one to three months of computing on an entire 512 core computer. The latest processors improve on this marginally. Ideally, we would like to reduce this by a factor of 10, not only to reduce delivery times on the results, but also to give us some latitude for re-runs.

It is inevitable that we frequently choose sub-optimal solutions because we cannot evaluate multi-disciplinary trade-offs between the requirements for various design solutions. In many cases, we *"know"* that a decision is wrong, but cannot offer the solid evidence. Sadly, electromagnetic hazards are what we describe as *"unwanted performance"* issues and is considered more as a constraint on design than a driver. On the other hand, the low observability and antenna installed performance issues are more *"wanted performance"* and are perceived more positively. The ability to do *good* analysis very fast would enable us to provide solid evidence quickly, and influence decisions more and more often and thus reach nearer optimal solutions. Cost benefits would arise from this, especially if we could prevent eventual re-design and re-work, which is exceptionally expensive.

However, there are many other opportunities, such as the use of virtual or augmented reality, that would substantially reduce design, manufacturing and modification times and costs, that we cannot reap advantage from as these would require near real-time simulations. More processors/cores, beyond a limit, not only does not help, it starts to hinder. This cannot be described as scalable computing.

3. EM Simulations on HPC Platforms

We have recently tested one of our major codes on a new parallel cluster based on AMD 16-core processors. Each node was equipped with a quantity of memory, and four processor sockets. Each processor was made up of two pieces of silicon, each with 8 cores. Even on a piece of silicon, the communications bandwidth was not uniform between all cores, and that between cores on different silicon and between processors on the node were different again. That between nodes was governed by the system interconnect, in this case, Infiniband at 40GByte/s. This architecture is shown in Figure 1.

Even more recently, the code has been tested on different, single substrate, 8-core devices. Figure 2 shows the effect on the number of iterations of the EM solver per hour by using more cores rather than processors to solve a given problem. In each case 24 cores were used, but using 2, 4 and 8 cores per processor on 12, 6 or 3 processors. Everything else remained the same. Thus, more cores does not lead to greater performance. In fact, much of the current HPC offerings may not be capable of delivering needed computing performance, though many-core processors are an attractive option to those who pay for the space, cooling and electricity consumption. The truth is, however, that if the computing power is needed, the bills just have to be paid.

[2]This is likely to be about the limit of time available. Testing in these cases is not an option as it would require an aircraft to be taken from the assembly line for a period of weeks for testing, and then requiring some degree of refurbishment with consequences for increased cost and delays in delivery.

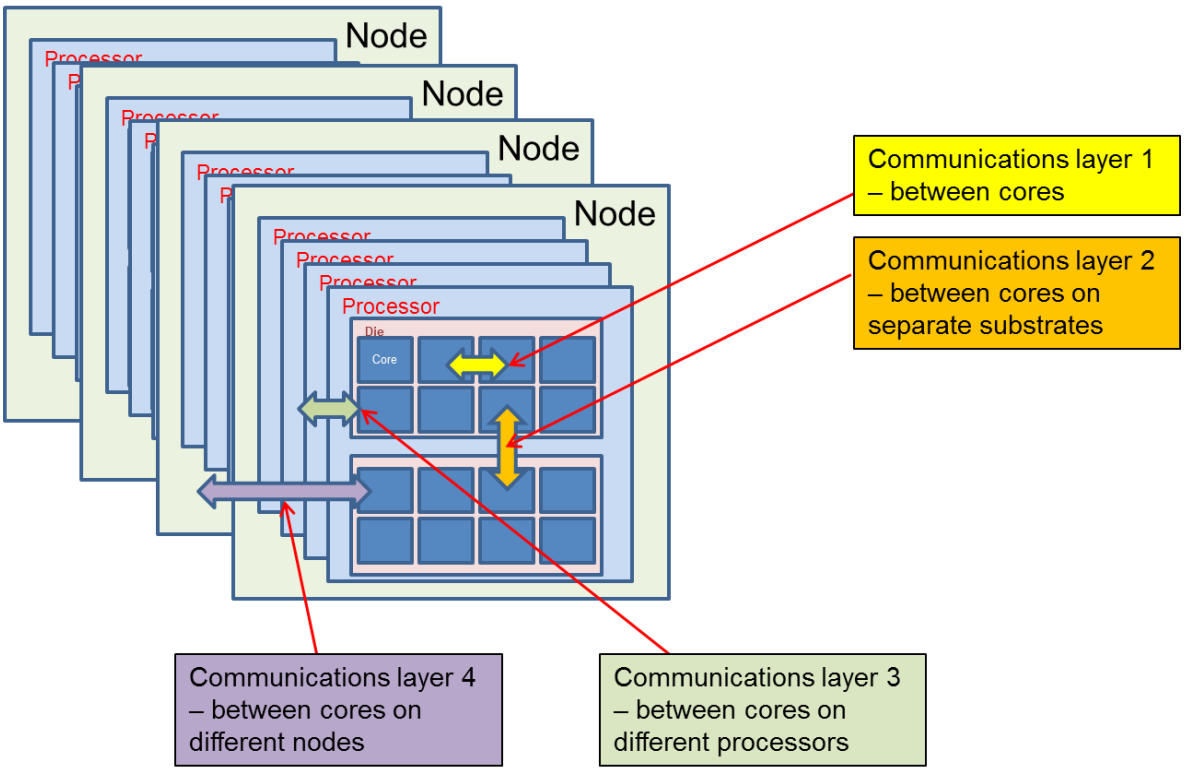

Figure 1. Typical communication hierarchy on modern HPC. Each layer has different bandwidth and latency. The concurrency programming model for layers 1, 2 and 3 is OpenMPI. For layer 4, it is MPI.

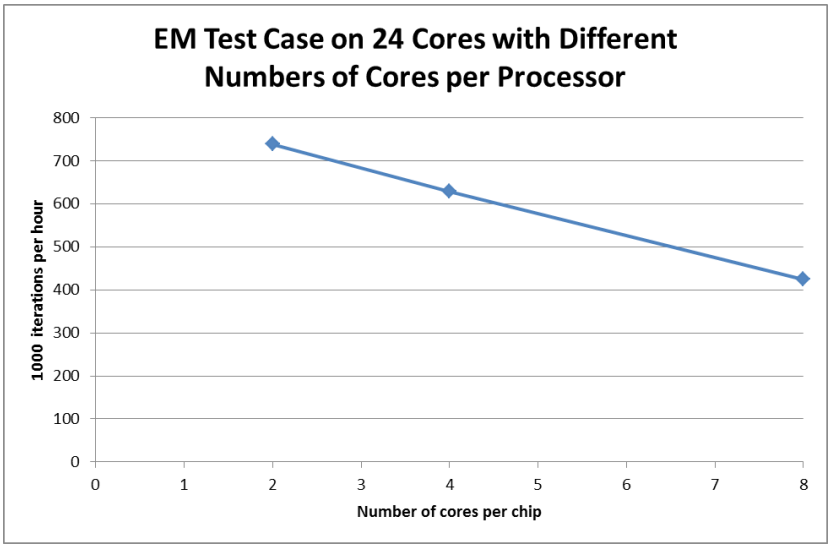

Figure 2. 24-core processing with various cores per processor.

This inhomogeneity introduces considerable complexity to understanding performance issues and load-balancing for optimisation. It also, usually, imposes at least two programming methods and libraries for message passing – OpenMP for inter-core/intra-node and MPI for inter-node communications – which itself adds complexity to understanding the parallel potential of processes and algorithms, as well as to implementation and maintenance, validation and documentation.

When we started out to use what was then known as *Massively Parallel Computing*, to run simulations of electromagnetic interactions, the introduction of Transputers was changing the world and offering affordable computer power of previously unavailable levels. This

was transformative in that what had not been economically viable suddenly became eminently realisable. At this time, the Transputer was thought of as the gateway to affordable High Performance Computing (HPC). The Transputer's role as a very good embedded device was a later and secondary consideration. occam was similarly considered. The demise of the Transputer had the unwanted side-effect of side-lining occam as a mainstream HPC programming language, which the developers of occam and its derivatives have been unable to counter. This is part of the reason we are in the HPC pickle that we are in.

4. What is needed for usable, scalable HPC

First, we are not asking for parallel computing nirvana: we are not asking that our poorly written serial codes should be able to run optimally on arbitrary parallel architectures and HPC implementations, nor even that our well sculpted parallel codes should always run optimally without some intervention. However, we should not have to implement multiple parallel programming paradigms within a single code and revisit the code for every new compute machine offered by the purveyors of hardware. We should certainly not have to abstract our physics to alternative numerical algorithms to try and demonstrate that we can benefit from HPC platform claims that they continue to adhere to Moore's law.

It is worth noting here that our primary code was, 10-15 years ago, considered to be embarrassingly parallel. It was originally written in Fortran, rewritten from the mathematical algorithms in occam and analysed using a CSP formal description. After the demise of Transputer, the occam code was translated into Fortran with some parts written in C to manage run-time array sizing. Since then there have been some minor modifications to optimise its performance for Power PC, Compaq Alpha, AMD and Intel processors in array sizes from 92 to 512 cores and increasing its performance in increments of approximately 10 times every 5 years.

However, for the next HPC upgrade, while it is easy to find solutions that offer nominal performance gains, it is somewhat harder to maintain the improvement in actual, delivered performance running our codes on our data. In short, while a typical simulation currently takes 3 days on 512 cores of five year old technology, it is hard to find a solution that will run the same problem in any less time, let alone say 12 hours; and impossible to find one that will solve the problem in 10 minutes or even 1 hour, no matter how much money might be available to be spent on the computer. Ten minutes is perhaps too severe a requirement, but surely, 1 hour is not. This is not scalable computing.

What I need is:

- occam or occam-*like* communications constructs[3] embodied in Fortran or C/C++. (Personally, I would prefer occam, but it is very hard to convince others to switch from a programming language they have come to love – it is an emotional thing.)
- A smart means of launching parallel processes[4], whether many of the same, some of a number of varieties, or different from each other.
- Programming, profiling, performance monitoring and debugging tools.

Some will tell me straight away that I can have all of these things now, but that isn't really quite accurate. I am sure that I could have much of what I ask for in tools that I must assemble and/or compile myself, and with no kind of support other than a personal telephone

[3]However, the constructs are not enough: an understanding of the underlying CSP process *model* (which does not mean an understanding of CSP itself) together with an appreciation of the occam parallel security rules concerning shared references and aliasing (automatically enforced by the occam compiler, but not by those for the other languages) is also needed.

[4]Looking further to the future, an ability for the *dynamic* construction of process networks – i.e. not just at start-up – may be useful.

call to someone I happen to know. Actually, I work in industry where the software must pass some formal stages of review to ensure it works, does what it is supposed to do, is supported to assist those using it now and indefinitely into the future, and will transfer to or be available on other hardware in case of disaster or replacement, etc...

Ideally, I would like hardware, or equivalent software implementation, that incorporates programmable interconnect logic that allows a number of defined point-to-point links to be established for a run-time, to remove the delays involved in arbitrary routing, plus the tools to devise the routing plan and to program it.

Why are the computer science departments around the country not researching in these areas? Why indeed do the most recent *strategy* reports [13,14] have nothing to say about these aspects of the problem of high performance computing?

The above is by no means a complete wish list: other aspects of HPC that should be seriously researched include:

1. Analogue computer co-processors and the language extensions to program them. Analogue computers are a faster and a nearer exact means of modelling many physical phenomena. The means to create them should be more accessible now, with lower noise and more stable operational amplifiers, than was the case in days of the greatest interest in them (1950s and early 1960s). Imagine what could be programmed with a large parallel array of conventional processors each with a digitally controlled analogue computer.
2. FPGA co-processors and the language extensions to program them. This is not such a pipe dream. Two such companies, AlphaData and Nallatech, have already contributed to a significant computing capability of this type at Edinburgh's EPCC. However, when we wanted to consider the technology for our HPC role, our code could not be benchmarked because of the difficulties in translating it to the different technology. Either computing hardware is usable or fated to disappear. The potential benefits of this approach are so significant, however, that more effort should be given to a coherent processor/FPGA co-processor programming language extension.

5. Conclusion

HPC (or supercomputing) has become a poor relation in the computing world. It is apparently not a large enough endeavour globally to command specific technology developments. Instead, it hangs on the coat tails of the consumer, even the games market. HPC platforms declare nominal performance from which few, if any, problems can possibly gain advantage and even fewer will ever be able to invest in the coding to do it.

The really disappointing thing is that UK led the world into the parallel computing age, both conceptually and practically. Short-sighted government and the computer industry let that lead disappear, leaving the world to the HPC cul-de-sac in which we now find ourselves. An extraordinary note is that we were aware of this at least as far back as 1995 and warned ourselves and any who would listen, at the *Crisis in HPC* workshop [15,16,17]. That crisis is upon us now.

Even so, the occam community still has some of the ingredients to mitigate the present problems, but has mostly lain down and merely purred since the demise of the Transputer, rather than boldly addressing the current issues. By all means carry on research into the intricacies of pure occam-ite programming, but also have a good look at how this elegant paradigm can be raised into useful functioning parallel computing tools despite the serious shortcomings of the hardware world.

References

[1] Charles A. R. Hoare. Communicating Sequential Processes. *CACM*, 21(8):666–677, August 1978.

[2] Charles A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[3] Andrew W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1997.

[4] Andrew W. Roscoe. *Understanding Concurrent Systems*. Springer, 2010.

[5] Steve Schneider. *Concurrent and Real-time Systems — The CSP Approach*. Wiley and Sons Ltd., UK, Baffins Lane, Chichester, UK, 1999. ISBN: 0-471-62373-3.

[6] Robin Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999. ISBN-10: 0521658691, ISBN-13: 9780521658690.

[7] Dick Pountain and David May. *A Tutorial Introduction to occam Programming*. McGraw-Hill, Inc., New York, NY, USA, 1987.

[8] M. D. May. CSP, occam and Transputers. In Ali E. Abdallah, Cliff B. Jones, and Jeff W. Sanders, editors, *25 Years of CSP*, volume 3525 of *Lecture Notes in Computer Science*, pages 75–84. Springer Verlag, April 2005.

[9] P. H. Welch and F. R. M. Barnes. Communicating Mobile Processes: introducing occam-π. In Ali E. Abdallah, Cliff B. Jones, and Jeff W. Sanders, editors, *25 Years of CSP*, volume 3525 of *Lecture Notes in Computer Science*, pages 175–210. Springer Verlag, April 2005.

[10] Peter H. Welch and Jan B. Pedersen. Santa claus: Formal analysis of a process-oriented solution. *ACM Trans. Program. Lang. Syst.*, 32(4):14:1–14:37, April 2010. `http://doi.acm.org/10.1145/1734206.1734211`.

[11] Peter H. Welch, Kurt Wallnau, Adam T. Sampson, and Mark Klein. To Boldly Go: an occam-π Mission to Engineer Emergence. *Natural Computing*, 11(3):449–474, September 2012. `http://dx.doi.org/10.1007/s11047-012-9304-2`.

[12] Richard Holland. FDTF analysis of nonlinear magnetic diffusion by reduced c. *IEEE Transactions on Antennas and Propagation*, 43(7), July 1995.

[13] Peter Coveney. Strategy for the UK research computing ecosystem. `http://www.clms.ucl.ac.uk/sites/default/files/ResearchComputing-glossy.pdf`, October 2011.

[14] Dominic Tildesley. A strategic vision for UK e-infrastructure. `https://www.gov.uk/government/publications/e-infrastructure-strategy-roadmap-for-development-of-advanced-computing-data-and-networks`, November 2011. Accessed: 2013-08-24.

[15] Preston Briggs. Crisis in HPC (post-workshop discussion), October 1995. `http://wotug.org/parallel/groups/selhpc/crisis/discussion/12229.html`.

[16] Chris P. Wadsworth and Roger G. Evans. Crisis in HPC (personal article in Flagship Bulletin), September 1995. `http://wotug.org/parallel/groups/selhpc/crisis/articles/flagship.html`.

[17] Crisis in HPC (summary of conclusions), September 1995. `http://wotug.org/parallel/groups/selhpc/crisis/conclusions.html`.