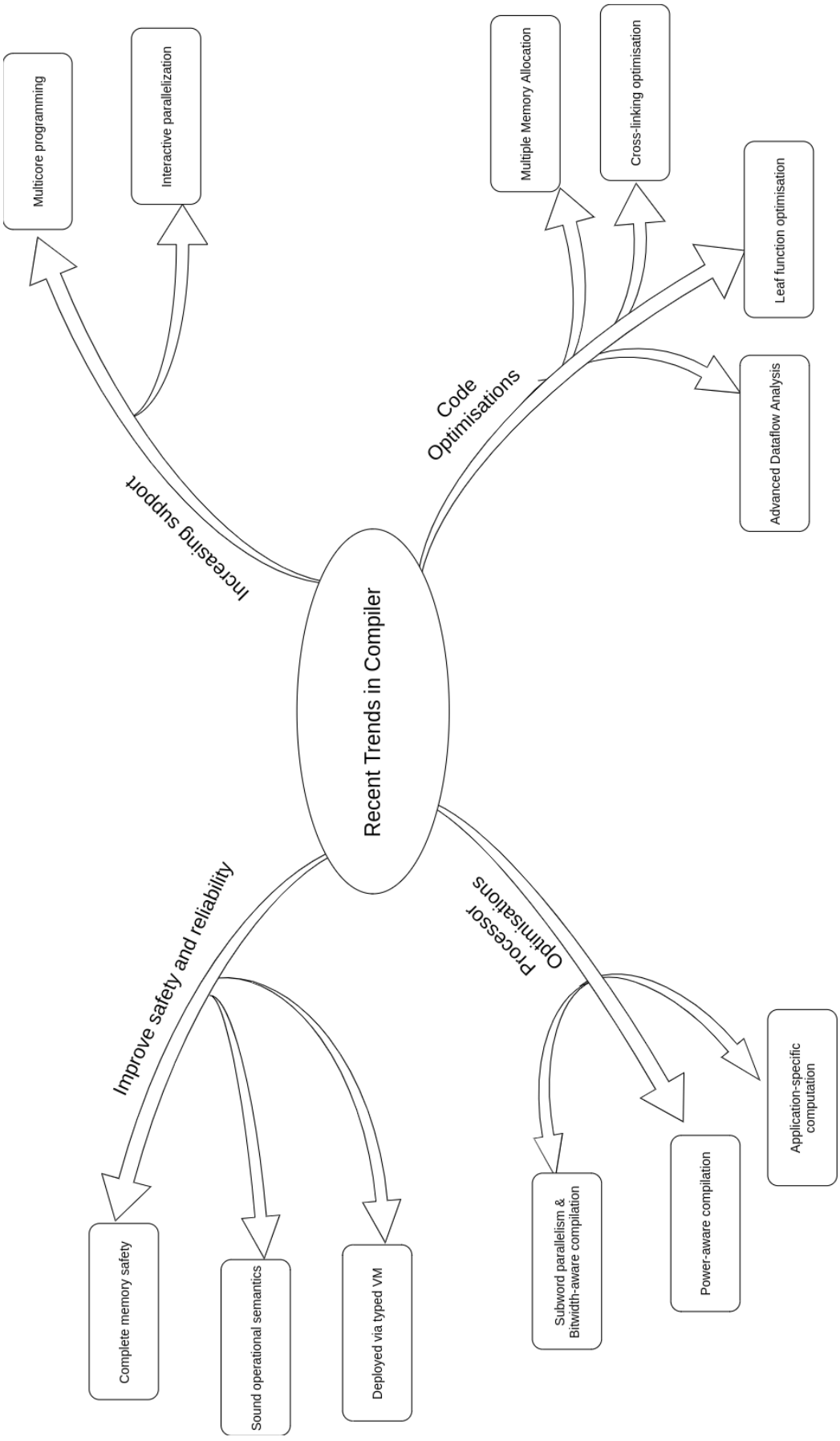
SSC Problem Based Learning -- Recent Trends in Compilers (Panel 1)

MIND MAP



Team:

Ritom Gupta
(PA25)

Rujul Walvekar
(PA30)

Pratik Gorade
(PA01)

T.Y B.Tech
CSE

SSC Problem Based Learning -- Recent Trends in Compilers (Panel 1)

SUMMARY

Recent trends in compiler can be categorised into 4 distinct viewpoints:

- ## Code optimisations:
    - **Advanced Dataflow Analysis:** During liveness analysis in conventional compiler analysis, the two common functions applied are the Gen and Kill function but most recent techniques in advanced compiler analysis also applies the Aliases in determining the liveness of the references at the exit of the block.

    - **Leaf function optimisation:** Leaf functions are those functions who do not directly call functions in a program. Leaf optimization achieves code reduction by creating these types of functions. When represented in a call graph, leaf functions forms the leaves of the call graph

    - **Cross-linking optimisation:** This method is commonly used in search engine optimization. Today, this method has also been applied in compiler optimization. Cross-linking can be applied locally or globally to functions that contain switch statements with similar tail codes. Since recent computer architectures are focused on code reduction, cross-linking has this as its major goal. When tail codes are spotted in a switch statement, cross-linking optimization algorithm is used to factor out this codes thereby reducing the actual size of the code

    - **Multiple Memory Allocation:** Multiple Memory Allocation (MMA) is one of the newest optimization techniques. It involves loading and storing each instruction in multiple registers. Microprocessors today use this method to reduce the code size. For example, in an ARM7 processor, the LDM instruction uses only sixteen bits to encode up to sixteen register loads (512 bits without the use of the LDM instruction).). Effective use of LDM and STM instructions in a ARM7 processor can save up to 480 bits opcode space

- ## Increasing support for:
    - **Multicore Programming:** One of the better and easier methods of doing this is to apply aggressive compiler optimization. A compiler that targets your processor and features advanced optimizations such as automatic vectorization, interprocedural optimization, and profile-guided optimization can substantially improve performance of your application. An absolute prerequisite for parallel optimization is highly tuned scalar performance. Parallel optimization must provide a 30% increase in performance over the scalar version of the application.

- **Interactive parallelization:** Interactive Parallelization Tool (IPT) assists domain-experts and students in efficiently parallelizing their existing C/C++ applications using any of the following parallel programming models: Message Passing Interface (MPI), OpenMP, CUDA, and hybrid programming

● Improving safety and reliability:

- **Complete memory safety:** Memory safety is the state of being protected from various software bugs and security vulnerabilities when dealing with memory access, such as buffer overflows and dangling pointers. For example, Java is said to be memory-safe because its runtime error detection checks array bounds and pointer dereferences. In contrast, C and C++ allow arbitrary pointer arithmetic with pointers implemented as direct memory addresses with no provision for bounds checking, and thus are potentially memory-unsafe.

- **Sound operational semantics:** The last few years have seen new language and compiler techniques (e.g. in the Cyclone, CCured, and SAFECode projects) that guarantee complete memory safety and sound operational semantics even for C and C++ programs. There is no longer any excuse for production C/C++ compilers not to provide these capabilities, at least as an option for security-sensitive software, including all privileged software.

- **Deploy via typed VM:** Furthermore, these capabilities can be deployed via a typed virtual machine that enables more powerful security and reliability techniques than with native machine code.

● Processor optimisations:

- **Power-aware compilation:** Reducing power and energy consumption is becoming increasingly important especially in embedded systems and application-specific processors. Compiler techniques to identify program regions that can be slowed down without significantly affecting the performance have been proposed. While executing these program regions, functional units can be slowed down by applying lower CPU voltages and frequencies (through dynamic voltage scaling).

- **Application-specific computation:** Instruction scheduling methods for DSP processors starting from simple list scheduling based heuristics to code compaction techniques, integer programming methods, combined register allocation and instruction scheduling.

- **Subword parallelism & bitwidth-aware compilation:** Many modern processors support instruction set extensions, e.g., multimedia extensions (MMX) and visual instruction set (VIS) for supporting subword parallelism which is exposed and exploited by many techniques. Subword parallelism is exposed through vectorization methods